

AD-A113 290

PHILCO-FORD CORP WILLOW GROVE PA COMMUNICATION SYSTEM--ETC F/G 17/2  
LONGBRAKE II.(U)

NOV 74 J M FORAN, J P JANOWSKI, G M KAUFMAN

DAAB03-74-C-0090

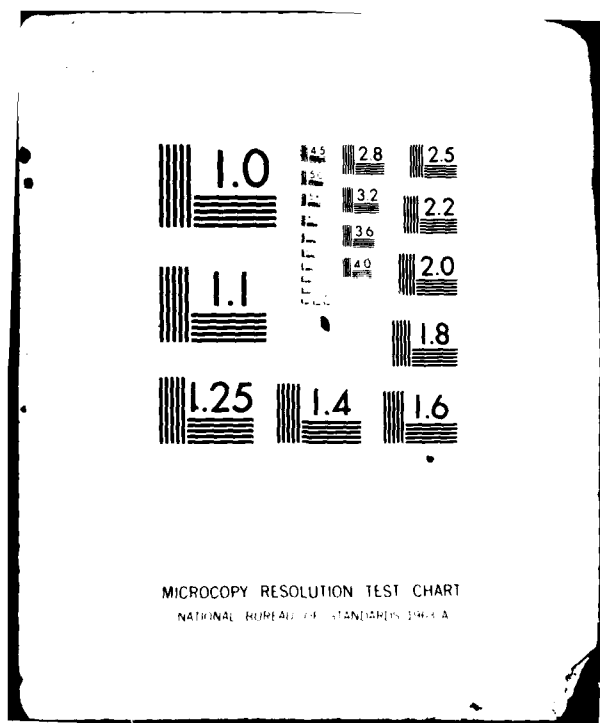
ML

UNCLASSIFIED

10/1  
Page



END  
DATE  
FILMED  
BY  
NTIC



Contract No. DAA803-74-C-0098

Cohn 13  
①

AD A113290

## Longbrake II

Final  
Report

**PHILCO**   
Philco-Ford Corporation

**DTIC**  
**ELECTE**  
APR 8 1982  
**S** **D**  
**H**

Prepared for  
NATIONAL SECURITY AGENCY  
9800 SAVAGE ROAD  
FORT GEORGE G. MEADE, MARYLAND

DTIC FILE COPY

NOVEMBER 1974

DISTRIBUTION STATEMENT A  
Approved for public release;  
Distribution Unlimited

82 03 22 144

FINAL REPORT

OF

LongBRAKE II

July 1973 through November 1974

Prepared for

National Security Agency

9800 Savage Road

Fort George G. Meade, Maryland

Contract No. DAAB03-74-C-0098

Submitted by:

John Welch  
John Welch

Approved by:

Charles S. Klayman and  
Charles S. Klayman  
Project Leader

Charles F. Tracher  
Charles F. Tracher  
Program Manager

22 November 1974

PHILCO-FORD CORPORATION  
Communication Systems Division  
3900 Welsh Road  
Willow Grove, Pennsylvania 19090

# ABSTRACT

Longbrake II is a 12 month program consisting of both study and hardware phases. The objective of the study is to develop an optimum Linear Predictive Coding vocoder system by optimizing the current real time algorithms used, and by the evaluation of additional LPC related techniques. The purpose of the hardware effort is to develop, test, and evaluate two Exploratory Development Models (EDM) of a speech compression system which incorporates the Atal and Markel approaches to adaptive linear estimation.

The work accomplished in the final quarter is summarized, a description of system performance over channels with bit errors is included, and a proposed ADM hardware design is described.

Accession For		
NTIS GRA&I		
DTIC TAB		
Unannounced		
Justification		
By		
Distribution/		
Availability Codes		
Avail and/or		
Special		

## TABLE OF CONTENTS

SECTION	PAGE
1 INTRODUCTION AND SUMMARY . . . . .	1-1
2 DISCUSSION OF THEORETICAL AND EXPERIMENTAL RESULTS	
2.1 General Description of the Final Longbrake System	2-1
2.2 Operational Programs to be Delivered with Equipment	2-8
2.3 Programs Developed During Last Quarter . . . . .	2-9
2.3.1 Encode and Decode . . . . .	2-9
2.3.2 Error Detection and Correction . . . . .	2-13
2.3.3 Final Voicing . . . . .	2-17
2.3.3.1 Computation of an Energy Reference Level	2-17
2.3.3.2 Description of Voicing Decision . . . . .	2-18
2.3.4 Real Time Operating System . . . . .	2-20
2.3.4.1 Description of Input-Output . . . . .	2-20
2.3.4.2 Back-to-Back System . . . . .	2-20
2.3.4.3 Transmit-Receive System . . . . .	2-21
2.3.5 Debug, Diagnosis and Test Program . . . . .	2-23
3 LONGBRAKE EDM EQUIPMENT . . . . .	3-1
3.1 Final Hardware Status . . . . .	3-1
3.2 Modem and Timing Interface Circuitry . . . . .	3-1
3.2.1 Functional Description . . . . .	3-1
3.2.2 Analyzer Output Buffer . . . . .	3-2
3.2.3 Synthesizer Input Buffer . . . . .	3-6
3.2.4 Synchronization . . . . .	3-9
3.3 Teletype Interface Circuitry . . . . .	3-13
3.3.1 Functional Description . . . . .	3-13
3.3.2 Printer Control Circuits . . . . .	3-14
3.3.3 Keyboard Control Circuits . . . . .	3-17
4 ADM DESIGN STUDY REPORT . . . . .	4-1
4.1 Consideration of Alternative Approaches . . . . .	4-1
4.1.1 Parallel LSI Microprocessors . . . . .	4-1
4.1.2 MSI/LSI Programmable Processors . . . . .	4-3
4.1.3 Pipeline Processors . . . . .	4-4

## TABLE OF CONTENTS (CONTINUED)

SECTION	PAGE
4.1.4 Recommended Approach - A Programmable Array Processor . . . . .	4-5
4.2 Description of Recommended Hardware . . . . .	4-6
4.2.1 Programmable Array Processor Concept. .	4-6
4.2.2 Programmable Array Processor Implementation . . . . .	4-12
4.2.2.1 Operand Memory . . . . .	4-14
4.2.2.2 Program Memory and Control . . . . .	4-19
4.2.2.3 Arithmetic and Logic Unit . . . . .	4-26
4.3 Programming the Processor . . . . .	4-32
4.3.1 Array Processing . . . . .	4-33
4.3.2 Efficiency of 3-Address, 3-Bus Structure	4-34
4.3.3 Memory Reference Flexibility Provided by Source-Source-Destination Concept. .	4-35
4.3.4 Special Feature of the PAP Instruction Set	4-36
4.3.5 Programming Loops on the PAP . . . . .	4-37
4.3.6 A Typical PAP Program . . . . .	4-41
5 PERFORMANCE EVALUATION TASK REPORT . . . . .	5-1
5.1 Description of Tests . . . . .	5-1
5.2 Test Results	5-2
5.2.1 Tests at 2400 Bits Per Second . . . . .	5-2
5.2.2 Tests at 3600 Bits Per Second . . . . .	5-3
5.2.3 Tests at 4800 Bits Per Second . . . . .	5-3
APPENDIX A - PRELIMINARY PAP INSTRUCTION SET . . . . .	A-1

## LIST OF ILLUSTRATIONS

FIGURE		PAGE
1-1	EDM Signal Processor - View 1 . . . . .	1-3
1-2	EDM Signal Processor - View 2 . . . . .	1-4
2-1	Longbrake System - Transmitter . . . . .	2-2
2-2	Longbrake System - Receiver . . . . .	2-6
3-1	Analyzer Output Buffer . . . . .	3-3
3-2	Synthesizer Input Buffer . . . . .	3-7
3-3	Synchronization and RCV Control . . . . .	3-10
3-4	Printer Control . . . . .	3-15
3-5	Keyboard Control . . . . .	3-16
4-1	Conventional Processor Organization . . . . .	4-10
4-2	Programmable Array Processor Organization . . . . .	4-11
4-3	Operand Memory Block Diagram . . . . .	4-15
4-4	Memory Operation Timing Diagram . . . . .	4-18
4-5	Instruction Timing Diagram . . . . .	4-20
4-6	Instruction Execution Timing Diagram . . . . .	4-21
4-7	Program Memory and Control Block Diagram . . . . .	4-25
4-8	Arithmetic and Logic Unit Block Diagram . . . . .	4-27
4-9	Significance of Instruction Fields . . . . .	4-38
4-10	Examples of Possible Program Loops . . . . .	4-40
4-11	Example Computation of Correlation Coefficients . . . . .	4-42
4-12	Coding for Computation of Correlation Coefficients . . . . .	4-43

## LIST OF TABLES

TABLE		PAGE
2-1	Modified Log-Area-Radio-Coding Parameters . . . . .	2-12
2-2	Allocation of E.D.&C Bits at 3600 BPS . . . . .	2-15
2-3	Allocation of E.D.&C Bits at 4800 BPS . . . . .	2-16

As a result of these efforts we have delivered hardware and software capable of providing full duplex digital speech communication at bit rates of 2400, 3600, and 4800 bits per second. The system can operate in either the Atal or Markel LPC modes and includes all necessary provisions for interfacing with modems at the 3 bit rates, for operating back to back, and for providing an analog test capability.

Error detection and correction is included when operating at 3600 and 4800 bits per second, and a software bit error generator is provided to test the effects of individual (non-burst) errors in the transmission system.

The delivered items also include a card reader for loading programs, a teletype interface and a debug and test software package for control and monitoring of programs, a full set of diagnostic software for hardware maintenance, and a FORTRAN based assembler to facilitate future software developments.

Figures 1-1 and 1-2 are photographs of one of the EDM signal processors delivered on this contract.

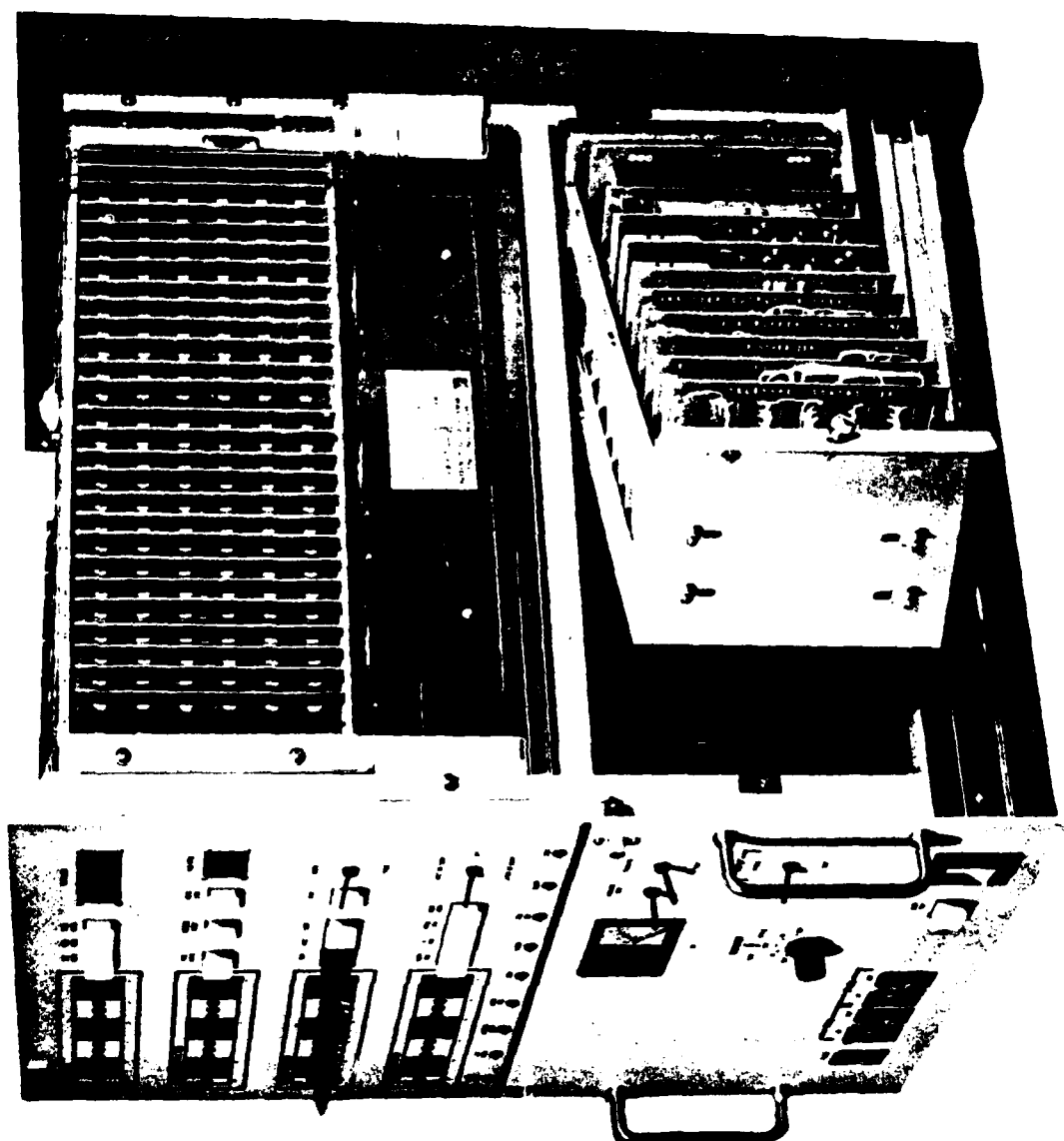


Figure 1-2. Longbrake EDM Signal Processor-View 2

## SECTION 2

### DISCUSSION OF THE THEORETICAL AND EXPERIMENTAL RESULTS

This report contains a description of the final version of the LONGBRAKE system which was developed by Philco-Ford for the Maryland Procurement Agency. The section contains descriptions of the overall system, the voicing algorithm, encode and decode, the error detection and correction system, the real-time operating system, and the DDT package.

#### 2.1 General Description Of The Final Longbrake System

Figure 1-1 is a block diagram of the final LONGBRAKE system as implemented on the LONGBRAKE EDM hardware. The speech signal is band-pass filtered with a gradual roll-off below 200 Hz and a sharp cutoff above 3200 Hz. It is then sampled at 6400 Hz. The speech samples are stored both into a pitch analysis buffer and a predictor analysis buffer at a 22.5 msec. frame rate. Single time-constant digital pre-emphasis with a treble boost above 700 Hz is applied to the data to be stored in the predictor analysis buffer.

For pitch extraction the data is low-pass-filtered by a cascade of 3 sample averaging filters, summing 5, 4, and 3 samples respectively, before being fed into the Average Magnitude Distance Function algorithm and the voicing detector. The voicing detector

uses an energy measure with a number of adaptive energy thresholds, zero crossing analysis, and the AMDF max-to-min ratio to make its decisions. The pitch and voicing rules apply smoothing and isolated error correction to the pitch and voicing values and, in the process, introduce two frames of delay into the corrected pitch values.

The predictive coding analysis uses either the Atal or Markel algorithms with a tenth order predictor.

The Matrix loading window has a fixed length of 102 samples which corresponds to 16 msec, but its starting point is varied to allow successive analysis windows to be separated by the exact multiple of the current pitch period which puts it closest to the beginning of the frame window. Since the range of possible phase variations is  $22.5 - 16.0 = 6.5$  milliseconds it is not always possible to separate the analysis windows by an exact multiple of the pitch period. In such cases the analysis window is placed as close as possible to the desired value, without moving it outside of the 22.5 msec. frame window.

Matrix loading uses full scaling of the input data and double precision accumulation of products. The Matrix values are block scaled to maintain 16 significant bits in the largest matrix element.

The RMS measurement is derived as a by-product of the computation of the main diagonal elements of the matrix, and the length of the RMS measurement window is always made equal to the largest multiple of the current pitch period  $\leq 92$ .

The matrix solution algorithm is a form of Cholesky decomposition with sophisticated block scaling to provide high accuracy

some protection against burst errors.

The final process in the analyzer is the conversion of the coded E.D.&C. output to a serial bit stream for transmission to the digital interface or modem.

Figure 1-2 is a block diagram of the receiver.

In the receiver program the serial bit stream is converted back to parallel, error detection and correction decoding is applied to the data, and then the data is decoded by the PARAMETER DECODING program.

The executive functions for the synthesis process are accomplished by PITSYN, the frame block to pitch block conversion and interpolation program. This program interpolates pitch, RMS, and reflection coefficients. Pitch and RMS are linearly interpolated, but RMS has an additional delay in interpolation at the onset of voiced sounds to increase the sharpness of the vocal attacks. The reflection coefficients are interpolated linearly, once per frame, at the frame mid point. Each particular pitch period is assigned that set of coefficients to which it is the closest in the frame. The PITSYN program also determines how many complete pitch periods can fit into the current frame and transmits that information to the control program which calls the synthesizer. Partial pitch periods are completed at the beginning of the next frame.

The subroutine which converts reflection coefficients to predictor coefficients is always called twice per frame. Once for the new set of reflection coefficients and once for the interpolated set. Therefore if we include the value at the beginning

(but not preemphasized) sampled input speech. When the switch is released, the systems will reinitialize in the current mode.

A program is also provided for introducing bit errors into the digital data stream. Various bit error rates can be obtained by simply changing one threshold in the program.

Finally programs are provided for converting the encode and decode functions from linear encoding to a form of log-area-ratio coding.

## 2.3 Programs Developed During the Last Quarter

### 2.3.1 Encode and Decode

Transmission of the LPC parameters at low bit rates requires that the parameters be encoded with great care. The coding system that is included in the LONGBRAKE EDM's involves different kinds of coding for the different parameters. The pitch period is encoded quasi logarithmically. Forty-eight possible delay values are encoded; ranging from 16 to 124. The values 16, 17, . . . , 31 are encoded as 1 through 16. The values 32, 32, . . . , 62 are encoded as 17 through 32. Finally 64, 68, . . . , 124 are encoded as 33 through 48. The encoding is accomplished in the pitch extraction routine itself, while decoding is done in PITSYN. Six bits are used to transmit the 48 pitch period values.

Voicing is encoded directly with one bit.

RMS is logarithmically coded from 1024 to 5 and linearly coded from 5 to 0. The logarithmic coding provides an increment of 1.748 db per step. The coding and decoding are accomplished by a tree search and a table-look-up respectively. The same 32 value coding table is used for both encode and decode resulting in a

centile points of the histograms and have used the following bit allocations for the 2400 bps and the 3600/4800 bps systems. At 2400 we transmit only nine reflection coefficients. The number of bits per coefficient is 6,6,5,5,4,4,4,4,3 for coefficients 1 through 9 respectively. At 4800 and 3600 bps the number of bits per coefficient is 6,6,5,5,5,5,4,4,4,4 for coefficients 1 through 10 respectively. At 2400 bps there is slight degradation audible in the speech quality as compared to full precision transmission of the parameters, but at 3600/4800 bps numerous A/B comparisons have been made with no indication of degradation due to this coding.

The second delivered option for coding the reflection coefficients is included as an overlay to the linear encode-decode scheme. This option is a slightly modified form of log-area-ratio coding. In this scheme the magnitude of each reflection coefficient can be considered to be transformed into a function Y by the following relationship.

$$Y = (\text{LOG}((F+RC)/(F-RC)) / \text{LOG}((F+MAX)/(F-MAX))) * NLEVEL \quad (1)$$

The values of MAX and NLEVEL differ for the different reflection coefficients and for the different coding rates and are tabulated in Table 2-1. If the parameter F is set equal to 1.0 this method is precisely log-area-ratio coding, since  $(1+RC)/(1-RC)$  is an area ratio. Log-area-ratio-coding results in nearly linear quantization for reflection coefficient magnitudes  $< 0.4$  and provides increasingly accurate quantization for the RC's as they approach unity in magnitude.

TABLE 2-1

## MODIFIED LOG-AREA-RATIO CODING PARAMETERS

I	2400 BPS		3600 & 4800 BPS	
	MAX	NLEVEL	MAX	NLEVEL
1	31/32	32	31/32	32
2	30/32	16	31/32	32
3	30/32	16	30/32	16
4	30/32	16	30/32	16
5	30/32	16	30/32	16
6	25/32	8	30/32	16
7	25/32	8	25/32	8
8	25/32	8	25/32	8
9	16/32	4	25/32	8
10	0	0	25/32	8

For RC's in the vicinity of 0.97, in fact, the quantization for high values can become so precise that it exceeds the inherent accuracy of the data. It is for this reason that we have introduced the factor F into the coding relationship. The first two reflections coefficients do attain magnitudes as high as 0.97. If F is unity, 6 bit log-area-ratio coding provides quantization precision for RC's near 0.97 that is equivalent to using 8 bit linear quantization. We have found, however, that the inherent accuracy of the RC's is such that there is no degradation from using 7 bit linear quantization of each reflection coefficient. By increasing F to 35/32 (1.09375) the quantization precision near 0.97 is reduced to a level that would be provided by using 7 bit linear quantization over the entire range. In the process, the precision for values near zero is increased from the linear equivalent of about 4 bits to about 5 bits. Hence by using 6 bit modified log area ratio coding with  $F=35/32$ , over a range of  $\pm$

without degradation is 61. Hence the number of bits left over for E.D. & C. is 20 at 3600 bps and 47 at 4800 bps.

At 3600 bps the 20 E.D. & C. bits are applied as shown in Table 2-2 The voicing bit is protected by majority 2 out of 3 logic. The four most significant bits of pitch are protected by a Hamming (8,4) code. The two most significant bits of the parameters RMS, K1, K2, K3, K4, and K5 are all protected by Hamming (8,4) or Hamming (7,4) codes.

The 4800 bps E.D. & C. scheme is summarized in Table 2-3. Voicing is protected by majority 2 out of 3 logic and all of the remaining parameters have their four most significant bits protected by Hamming (8,4) or (7,4) codes. Both Hamming codes are capable of detecting and correcting one error in the 8 or 7 bit code words. The Hamming (8,4) code is also capable of detecting, but not correcting two bit errors. All parameters which are protected by a Hamming (8,4) code are arranged so that a two bit error causes the value of that parameter which was used in the previous frame to be repeated in the current frame.

In the process of converting the parallel code words to a serial bit stream we have been careful to separate the bits of a given code word by as much as possible within the frame to provide some protection against burst errors.

The programs which implement the E.D. & C. algorithms have been written to maximize modularity and clarity but the programs which actually do the bit packing and calling of the Hamming coding subroutines are relatively long and cumbersome, so that changing the allocations can require a substantial amount of reprogramming.

Table 2-2

## Allocation of E.D.&amp;C Bits at 3600 bps

<u>Parameter</u>	<u>Number of Coding Bits</u>	<u>Number of Bits Protected</u>	<u>E.D.&amp;C Bits Used</u>	<u>E.D.&amp;C Algorithm</u>
Voicing	1	1	2	Majority 2 of 3
Pitch	6	4	4	Hamming (8, 4)
RMS	5	2	4	Hamming (8, 4)
K2	6	2		
K1	6	4	4	Hamming (8, 4)
K3	5	2	3	Hamming (7, 4)
K4	5	2		
K5	5	2	3	Hamming (7, 4)
K6	5	2		
K7	4	0		
K8	4	0		
K9	4	0		
K10	4	0		
			<u>Total 20</u>	

Table 2-3  
Allocation of E.D.&C. Bits at 4800 bps

<u>Parameter</u>	<u>Number of Coding Bits</u>	<u>Number of Bits Protected</u>	<u>E.D.&amp;C Bits Used</u>	<u>Algorithm</u>
Voicing	1	1	2	Majority 2 of 3
Pitch	6	4	4	Hamming (8, 4)
RMS	5	4	4	"
K1	6	4	4	"
K2	6	4	4	"
K3	5	4	4	"
K4	5	4	4	"
K5	5	4	4	"
K6	5	4	4	"
K7	4	4	4	"
K8	4	4	3	Hamming (7, 4)
K9	4	4	3	"
K10	4	4	3	"
<u>Total</u>			<u>47</u>	

### 2.3.3 Final Voicing

#### 2.3.3.1 Computation of an Energy Reference Level

Philco-Ford has developed and implemented a voicing decision for the Longbrake system which is based primarily on the measurement and comparison of low frequency energy values computed on a frame-by-frame basis. Due to the nature of speech excitation, voiced sounds have a generally higher low-frequency energy content than unvoiced sounds. This pattern can be greatly obscured, however, by changes in the overall amplitude of the speech signal.

The key to an energy-based voicing decision is the choice of an energy reference level (or levels) which will enable the voicing decision to be independent of changes in overall speech volume. The energy reference level associated with an earlier Longbrake voicing decision was updated at two second intervals and consisted of the maximum low frequency frame energy measured during the preceeding two second intervals. The profile described by this reference level was discontinuous and varied with the phase relationship between the speech signal and the arbitrary two second energy intervals. This arbitrary nature of the energy reference level led to a generally non-repeatable voicing decision. Also, the rigid two-second "holding" period limited the adaptability of the reference level to rapid inflections and weak trailing voiced syllables.

A combination of experimentation and induction led to the energy reference profile currently used in the Longbrake voicing decision. The profile consists of linearly decaying peak detection

of frame-by-frame energy values. The time constant is such that complete decay occurs in approximately five seconds. Two quantities are stored by the machine language routine which computes the energy reference level for each frame. The first is the reference level associated with the previous frame and the second is a step size. If the reference level is less than the current energy measurement, the reference level is reset to this value, otherwise it is unchanged. If the reference level has been replaced by the current energy measurement, the step size is reset to the appropriate fraction of this quantity. If the reference level has not been replaced, the step size remains unchanged. The step size is finally subtracted from the energy reference level. Thus, when the frame energy level rises above the reference level, the reference level immediately follows upward to the peak. As the energy decreases, the reference level decreases according to the current step size which produces a five second decay. The peak detection nature of this energy reference profile leads to a repeatable voicing decision and the built-in decay allows the voicing decision to follow falling inflection and low-volume voiced syllables. If a silent period of sufficient duration occurs, the reference level will completely decay, but will immediately respond to the first utterances.

#### 2.3.3.2 Description of Voicing Decision

The voicing decision is a logical flow routine using measured properties of the speech waveform and a series of thresholds. Four energy dependent variables are derived from the energy reference

level described above. One of these establishes the level of a dither signal, and the other three are used as thresholds for energy comparisons.

A dither signal, proportional to the energy reference level, is added to the lowpass filtered input signal as a zero crossing count is performed. If the resulting zero crossing count exceeds a fixed threshold, an unvoiced decision is made. Thus, the dither signal forces the very low energy intervals, which are generally associated with background noise, to be unvoiced. In addition, high frequency strongly unvoiced sounds will be detected as unvoiced simply because of their high zero crossing content. If the zero crossing count threshold were lowered to the point where it could properly detect all unvoiced sounds, it would improperly detect many voiced sounds, therefore it is not used for discrimination of marginally voiced sounds.

The difficult marginal voicing decisions use only the lowpass filtered energy computation and the maximum-to-minimum ratio of the normalized AMDF function. (The dependency of the voicing decision on the AMDF max.-to-min. ratio is, however, very weak, and little performance would be lost by completely eliminating it as a factor). High lowpass energies and high AMDF ratios both tend to indicate a voiced decision. Two fixed AMDF ratio thresholds separate the AMDF ratio measurements into three regions. Corresponding to each of these regions is a different energy threshold expressed as a fraction of the energy reference level. AMDF ratio values in the highest AMDF ratio region result in the lowest fractional energy threshold. For each frame, the current AMDF ratio is

detected to be in one of the three regions mentioned above. If the lowpass energy then exceeds the fractional energy reference threshold corresponding to that AMDF ratio region, a voiced decision is made; otherwise the frame is labeled as unvoiced. Since the energy thresholds are computed relative to the energy reference level, the voicing decision is made independent of the overall speech amplitude.

#### 2.3.4 Real Time Operating System

##### 2.3.4.1 Description of Input-Output

Incoming data from the A/D converter is stored in two adjacent buffers each of which contains 145 locations. At a sampling rate of 6400 Hz. and a frame time of 22.5 milliseconds, there are 144 samples per frame. While one of the input buffers is being analyzed, the other is being filled with continuously clocked input speech data.

Output speech data is stored in three adjacent buffers, each containing 144 locations. These buffers are filled by the synthesizer in pitch period blocks of samples while the output speech is continuously clocked out to the D/A converter.

##### 2.3.4.2 Back-to-Back System

This system is self-contained and performs linked analyses and syntheses. The real time operation is governed by the D/A interrupt. This interrupt occurs every 144 sample times and initiates a sequence of events. During the servicing of this interrupt the A/D input buffers and the analyzer references to that buffer are switched. Since the input buffers contain 145 locations, there will be an unused location in each frame for the back-to-back mode. The output buffers are also switched under this interrupt.

Finally a flag is set before leaving the interrupt service routine.

When the program is started it goes through an initialization procedure, and into an idling "spin" loop awaiting the flag which is set in the interrupt routine. When the flag is detected, it is reset and an analysis is begun. The analyzed frame always contains 144 samples. Upon completion of an analysis, data is transferred and a synthesis is performed. Syntheses are always targetted at producing 144 samples per frame. Upon completion of synthesis, the spin loop is again entered and the process repeats itself when the interrupt flag is detected.

#### → 2.3.4.3 Transmit-Receive System

In the full-duplex transmit-receive system, two constantly clocked bit streams transfer data from the local analyzer to the remote synthesizer and from the remote analyzer to the local synthesizer. In this system there are four timing sources versus one in the back-to-back system; and three interrupts govern the real-time operation. The four timing sources consist of two internal processor clocks and two modem clocks. The modems clock bits in and out of specialized modem interface hardware. The three interrupts are a transmit interrupt, a receive interrupt, and a D/A interrupt. A transmit interrupt indicates that the transmitter interface buffer has been exhausted. Under this interrupt, this buffer is refilled with analysis data. In addition, the input buffers are switched and an analysis flag is set. Due to drift between the processor and its transmitter clock, there may be either 143, 144, or 145 samples inputted between transmitter

interrupts. These departures from 144 are linked into the analyzer which adjusts its frame size accordingly. A receive interrupt indicates that the receiver interface buffer has been filled. During the interrupt servicing routine, this buffer is emptied into an area of memory reserved for synthesis data and a synthesis flag is set. The number of D/A conversions since the previous receiver interrupt is computed and this number is the target number of samples per frame for the synthesizer (143, 144, or 145). The D/A interrupt occurs every 144 samples and switches the output buffers.

The analysis, and synthesis flags are stored in a list in their order of occurrence. After an initialization procedure, the program spins while checking for flags. When a flag is detected it is deleted from the flag list and the appropriate analysis or synthesis operation is performed. Upon the completion of this operation, the spin loop is again entered. If a flag remains on the list it will be immediately serviced. If not the program will spin until a flag is set.

Until synchronization occurs between transmitter and receiver, only transmit interrupts will be generated, and only analyses will be performed. During this time, the D/A is squelched to silence.

Simulated transmit-receive operation can be performed by one processor without a modem. This is done by moving a front panel switch from Operate to the Test position. The bit stream is routed directly from transmit buffer to receive buffer and is clocked by a second processor clock.

A final real-time mode is the Analog Test mode selected by

a processor front panel switch. While this switch is activated, the program will transfer data from the input buffers directly to the output buffers using only the D/A interrupt for buffer switching. When the switch is released, the program will restart. This mode does not provide an ideal A/B comparison capability because the analog test path introduces deemphasis into the signal with no corresponding preemphasis. Hence it does not provide an accurate timbre match to the processed speech which has both preemphasis and deemphasis.

#### 2.3.5 Debug, Diagnosis, and Test Program

A software debug, diagnosis, and test program was written for the EDM processor which, together with the TTY unit, greatly facilitates the testing and debugging of programs on the Philco equipments. The program occupies approximately 1300 (octal) locations and has been written to reside at the upper end of memory.

In essence, the DDT program is similar to that used on other machines. It provides the ability to enter data or instructions (in various formats) into memory or to examine the contents of memory and print out those contents (again, in various formats).

The DDT program is essentially an I/O control program which employs various command structures for its operation. The commands used by the programmer to run DDT are expressions, formats, and control. The DDT software makes use of sequences of parameters, called expressions, which are evaluated algebraically or logically and converted to internal values. These values are used either to

address locations in the machine or to enter data into the machine. The format commands enable the user to select the type out of the machine to be printed in any one of five modes: unsigned octal integer, unsigned decimal integer, unsigned base R integer, signed decimal fraction, or two ASCII "Test" characters. Finally, the control commands provide the user the ability to clear memory, to search memory (for a match or no match), to execute a program, and to enter and remove breakpoints in a program.

## SECTION 3

### Longbrake EDM Equipment

#### 3.1 Final Hardware Status

The hardware debugging of the two Longbrake EDM equipments was completed during this reporting period. Most hardware problems were associated with the main memory where many defective and marginal integrated circuits were replaced. A unique problem was exhibited during the early debugging of the second unit in that approximately thirty missing, broken or shorted connections were found in the Gardner Denver Wiring. The teletype interface was checked with a Model 33 ASR Teletype and was used extensively with the DDT software in debugging the hardware and other software. After initial debugging the entire diagnostic software package verified proper operation of both equipments. The EDM units were tested also as an operational LPC system. Full duplex operation was checked at 2400 and 4800 BPS through external modems that provided external timing and exercised the data interface hardware. Full duplex operation at 3600 BPS was also checked with the units directly interconnected while using an external timing input. All documentation was updated and submitted with the units for final QA inspection.

#### 3.2 Modem and Timing Interface Circuitry

##### 3.2.1 Functional Description

The data interface hardware operates in concert with the I/O

functions of the EDM to reformat data between the parallel structure of the processor and the conventional serial format used in data transmission equipment. Functionally the interface provides data buffering and synchronization capability, processor interrupt generation, and standard MIL-STD-188 interfaces for the external data and timing signals. The interface circuitry is located on four PW cards housed in the analog card cage of the EDM. All signals and controls necessary for full-duplex operation are brought out on rear connector J3. The interface circuitry provides for flexibility in the selection of data rates and coding formats. No internal wiring modifications are required for different rates or formats as the countdown from the external timing input to the frame rate can be selected from either control switches mounted on one of the PW boards or remotely via control lines or rear connector J3.

The hardware is proportioned between the Analyzer Output Buffer, the Synthesizer Input Buffer 1 and 2, and the Analyzer/Synthesizer Buffer Control. The following paragraphs describe the operation of the analyzer output buffer, the synthesizer input buffer and the synchronization circuits.

### → 3.2.2 Analyzer Output Buffer (Figure 3-1)

The analyzer output buffer, contained on the PW board of the same name, provides the interface and storage for transferring data from the 16 bit parallel format of the machine to serial data for output to an external device. The buffer accepts successive 16 bit words from the processor, assembles them into a serial frame of data, and outputs the frame at a rate determined by the external

clock.

The external timing signal is brought into a MIL-STD-188 receiver from the rear connector and a strapping option is supplied to select output data to coincide with either edge of the data clock as required by the external system. The data output is provided to the rear connector from a MIL-STD-188 output driver. Selection gates on the Analyzer/Synthesizer Control card allow the simultaneous routing of data and clock from either external or internal paths. A front panel Operate/Test switch in the test position will select the internal timing mode which in concert with the data selection gate and some software changes allows the interface to be used in the back-to-back mode.

The number of bits per frame is selected by the frame size switches on the Analyzer/Synthesizer Control board. The frame length can be set from 1 to 127 by converting the number of bits per frame to a binary number then setting the switches corresponding to a '1' to the ON position. Optionally, all switches can be set to the OFF position and the frame length can be controlled remotely through the rear connector by providing a ground for each input line corresponding to a "1" in the desired binary number.

The data buffer is a 128 bit parallel-to-serial converter divided into 8 segments of 16 bits each. Since both the frame counter and the parallel-to-serial converter are clocked by the same timing signal, the frame counter is effectively counting the number of bits which have been shifted out in serial from the parallel-to-serial converter. When a complete frame of data has been shifted out, a frame counter output will initiate an interrupt of the pro-

Important

cessor, labeled TX INTSTRT, and will also reset on an 8 bit shift counter that generates loading pulses for the data buffer. The interrupt (level 4) causes the processor to execute an interrupt routine which reloads the parallel-to-serial converter with a new frame of data by executing successive COD instructions.

The first COD (connect output data) instruction to be executed loads 16 bits of data into the output register of the processor, then generates a data sent pulse (CH1 DSL in the block diagram). The right most AND gate in the figure will be enabled to allow the data sent pulse to be gated to the load control of the right most 16 bit parallel-to-serial converter, causing 16 data bits from the output register to be loaded in parallel. The trailing edge of the data sent pulse clocks the 8 bit shift counter which enables the AND gate to the second parallel-to-serial converter and disables the first AND gate. The next COD to be executed therefore loads 16 bits into the second parallel-to-serial converter. This process continues until a complete frame of data has been loaded. For example, if the frame length is set to 54 bits, then 4 COD instructions would have to be performed to reload the data buffer. Actually, 4x16 or 64 bits of data would be loaded with the last 10 bits being nonsense which would not be shifted out before another reloading would occur. The sync or framing bit is the last bit of each frame. No hardware is made for inserting this framing bit and thus this must be performed in the software control routine.

Since the parallel-to-serial converter is being continuously shifted to produce a serial bit stream, the processor has one bit

time to load a new frame of data into the parallel-to-serial converter when depleted. At 4800 BPS data rate, this gives the processor 208 microseconds to load a frame.

Assuming that a new frame of data has been properly loaded, the first serial rate clock transition after the interrupt shifts the first bit of the frame into a one bit buffer register which connects through a MIL-STD-188 interface circuit to the rear connector. When the last bit of the frame has been clocked into the one bit buffer register, the interrupt occurs and the loading process is repeated.

### → 3.2.3 Synthesizer Input Buffer (Figure 3-2)

The synthesizer input buffer stores serial data received from an external device, in a serial-to-parallel converter then transfers data to the processor in groups of 16 bits. The buffer is loaded at a rate determined by the external receive timing and unloaded by issuing successive CID instructions from the processor to the control circuitry. The synthesizer input buffer is contained on Synthesizer 1 and Synthesizer 2 PW boards.

The data and timing inputs are brought into MIL-STD-188 receivers from the rear connector and a strapping option is provided to select the proper clock phase for center-sampling the input data into the buffer. Selection gates on the Analyzer/Synthesizer Control board allow the simultaneous selection of data and clock from either the external or internal paths. When the front panel Operate/Test switch is in the test mode, the analyzer output buffer is internally connected to the synthesizer input buffer and sample rate counter 2 (SRC2) is provided as a clock to

both buffers. Sample rate counter 2 must thus be enabled by the software via a COD instruction.

The serial rate clock is counted down to the framing rate by the receiver frame counter which can accommodate frame lengths from 1 to 127 bits. The frame size selection is made by either switches on the Analyzer/Synthesizer Control board or remotely via a rear connector. These controls are common with the analyzer frame counter and thus one setting controls both counters.

The data buffer is 128 bits long which can store eight 16 bit words. ① Data is clocked into the buffer at the serial rate and transferred out of the buffer and into the processor on the CH1 data bus in groups of 16 bits. ② The serial data received is shifted into the synthesizer input buffer and the number of data bits accumulated is tallied by the receiver frame counter, which provides and end of frame signal when a complete data frame has been assembled. ③ The end of frame signal triggers an interrupt (level 5) to the processor to load the data for the synthesizer program. ④ The interrupt causes the processor to execute a routine consisting of successive CID instructions for loading data through the CH1 bus. ⑤ The first CID instruction will load the last 16 bits in the buffer into the processor in a parallel transfer. A CH1 data received (CH1DRL) signal will be provided by the processor and this will cause a parallel transfer in the data buffer, shifting all data by 16 bits. The register that contains the last 16 bits received then contains the previous 16 bits. ⑥ Additional CID instructions will be issued until all data within the frame is loaded into the processor.

The synthesizer buffer can thus shift data serially for loading or in 16 bit jumps in the reverse direction for unloading.

Referring to Figures 3-2 and 3-3, the REV REG MODE controls either serial or parallel shifting of data in the buffer. The RX REG CLK signal which serially shifts the buffer is a retimed pulse formed near the timing transition that center-samples the input data. When a frame of data is accumulated, the interrupt to the processor (labeled REC INT START) sets the REV REG MODE to the parallel transfer mode. Each CH1 data received (CH1 DRL) will shift the buffer 16 bits so the words load to the processor in reverse order. <sup>⑤</sup> Notice that the first bit of the frame is in the last group transferred to the processor. Care must be exercised when loading and unloading data to the analyzer and synthesizer buffers to maintain the proper order of data. <sup>⑥</sup> The unloading process, as in the analyzer, must be completed within one bit time following the interrupt. The next timing transition will reset the RCV REG MODE for serial transfer and the first bit of the new frame will be shifted into the buffer.

#### 3.2.4 Synchronization (Figure 3-3)

The synchronization circuits establish frame sync between the synthesizer program within the processor and data received from the remote analyzer. The frame intervals of the data from the analyzer can be identified by the occurrence of a sync bit in each frame. The sync bit is located in the final bit position of the data frame and alternates in level between adjacent frames. The synchronization circuits must locate the sync bit in the serial

input data stream and then phase lock the receiver frame counter so that the end-of-frame decode from the counter occurs coincident with the reception of the sync bit.

The sync circuits have three modes of operation. The first mode is to detect and "lock on" to a possible sync bit position in the received data stream. The second mode is to verify the presence of the alternating pattern during subsequent frames and to finally declare sync if verification is made. The third mode is to monitor the sync bit for loss of sync. If verification is not made in the second mode or if a loss of sync is detected in the third mode, the control returns to the first mode and the cycle is repeated to re-establish correct synchronization.

① The sync circuits (Figure 3-3) contain an exclusive-OR com- ①  
parator for comparing input data to a '1-0' memory which stores  
the expected state of the alternating sync bit. ② The end-of-frame  
decode from the frame counter acts as a strobe for sampling the  
comparator once per frame to the sync counters. ③ One counter counts  
correct sync bit comparisons and the other tallies errors. ④ In the  
initial out-of-sync state, the search control flip-flops hold the  
sync strobe continually enabled until a logic one input data is  
received. ⑤ The '1-0' memory is held to the logic one state and the  
enabled sync strobe holds the frame counter in the reset state.  
Each input logic zero will cause an incorrect comparison to the  
'1-0' memory and the counter for correct comparisons will be reset.  
As long as input data bits are logic zeros, the sync circuits will  
remain in this "idle" state. ⑥ When the first logic one input is  
received, the exclusive-OR comparator will indicate a correct

comparison. <sup>(1)</sup> The search control flip-flops will reset, the counter for correct comparisons will step one count, and the '1-0' memory will toggle. The resetting of the search control flip-flops allow the sync strobe to go inactive and the frame counter to stop. <sup>(4)</sup> The sync circuits are now in their second mode of operation and one frame later, the frame counter will generate an end-of-frame signal or sync strobe to check the status of the suspected sync bit. <sup>(2)</sup> If a correct comparison occurs, the correct comparison counter will step and the '1-0' and the memory will toggle. <sup>(3)</sup> The frame counter will continue to run to check the same bit position of the following frame. If, however, the comparator indicated an incorrect level for the suspected sync bit, the correct comparison counter would be reset and the search control flip-flops would be set to return the sync circuits to the initial searching conditions.

<sup>(5)</sup> Sync will be declared when <sup>(6)</sup> 8 consecutive correct comparisons occur. At this time the sync status flip-flop is set to the "in-sync" state and a level 5 interrupt is generated to the processor. <sup>(7)</sup> The sync status flip-flop appears on bit 8 of the input status word and can be checked by the processor through a CIS (check input status) instruction. The sync bit is the last bit received in a frame so when the interrupt occurs, the processor can input a complete frame of data from the synthesizer input buffer by generating a series of CID instructions in an interrupt routine.

Once sync is declared, the sync circuits enter the third mode of operation where the sync bit is monitored to detect a loss of synchronism between the receiver frame counter and the input data frames. Each time the sync strobe becomes active to check the com-

①

parison of the sync bit to the '1-0' memory, either the correct comparison counter will step or the counter for incorrect comparisons will be advanced. When the sync strobe becomes active both counters are primed for counting on the next clock input. However, if an incorrect comparison occurs between the input data bit and the '1-0' memory, the load input to the "right" counter will become enabled and this counter will be reset while the "wrong" counter will step one count. If a correct comparison occurs, the count enable input to the "wrong" counter will be inhibited while the "right" counter is stepped one count. The in-sync condition of the sync status flip-flop alters the correct comparison counter such that the function of the counter becomes to reset the "wrong" counter whenever two consecutive correct comparisons occur. This incorrect comparison or error counter will reset the sync status flip-flop to the out-of-sync state if 8 errors are accumulated without 2 consecutive correct comparison occurring. Simultaneous with the resetting of the status flip-flop for an out-of-sync condition, the search control flip-flops will be set to return the sync circuits to the initial searching conditions. The interrupts to the processor from the receiver circuits will be disabled until synchronization is again established.

P.

### 3.3 Teletype Interface Circuitry

#### 3.3.1 Functional Description

The Longbrake processor contains a Teletype Controller card which enables it to interface with a Model 33ASR. This Teletype has a character format of 11 bits, consisting of one start bit (a space), 8 data bits which use the USASCII standard code, and

2 stop bits (marks). Each bit has a duration of 1/110 second.

The processor interfaces with the controller card on Input/Output Channel 3, by means of 8 output data lines from the output register in the processor, a data needed input signal to the processor, a data sent output from the processor, 8 input data lines from the controller, a data available input to the processor and a data received output from the processor. The controller interfaces with the ASR over a pair of wires to the ASR printer and a pair from the keyboard through connector J1 on the rear of the processor.

Any program written to interface with a Teletype through the controller card must first sense the status of the controller circuits before sending data to or receiving data from them. To send a character, the processor must first sense the data needed signal from the controller, using the CIS instruction. Only when this signal is ON should the processor perform a COD instruction to transfer 8 bits of data to the controller. To receive a character, the processor must first sense the data available signal from the controller, using the CIS instruction. When this signal is ON the processor can then perform a CID instruction to move 8 bits of data into the processor.

Refer now to the Teletype controller block diagram for a detailed discussion of the operation of the controller. The printer control circuits is shown in Figure 3-4 with the keyboard circuits shown in Figure 3-5.

### 3.3.2 Printer Control Circuits

When no character is being sent to the printer, the printer

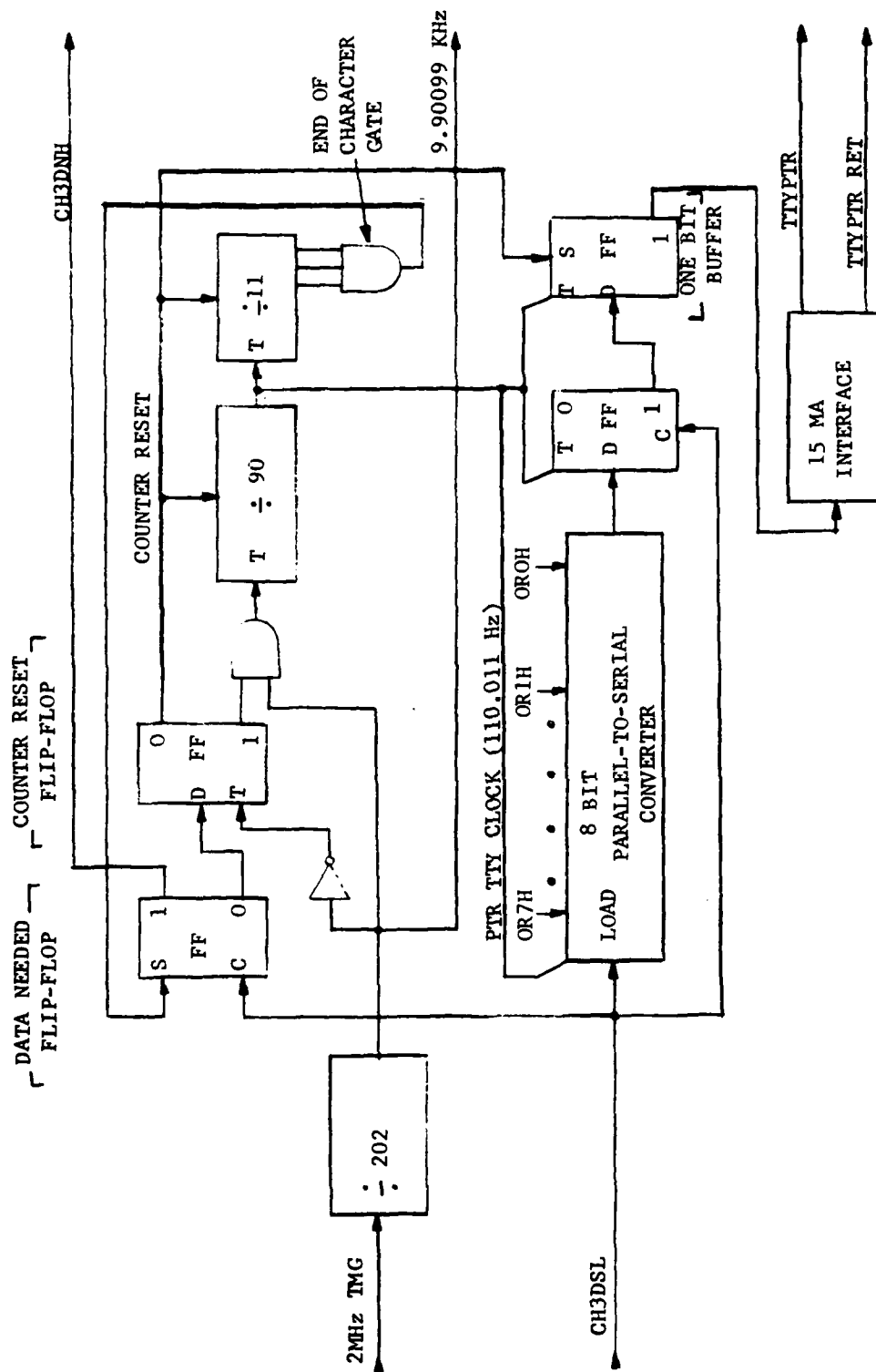


FIGURE 3-4. Printer Control



circuits are in their steady state or rest state, ON, and the one bit buffer is held in the set state sending a constant MARK to the printer.

The processor can now execute a connect out device (COD) instruction, which loads the output register in the processor with data and also generates a data sent signal (CH3DSL in the block diagram) which loads the parallel to serial converter, clears the space flip-flop, and clears the data needed flip-flop.

Clearing the data needed flip-flop causes the counter reset flip-flop to be set synchronously with a 9.9 KHz timing signal, allowing the divide by 90 counter to generate the PTR TTY clock.

This clock shifts a 10 bit shift register consisting of the 8 bit parallel to serial converter, the space flip-flop, and the one bit buffer register. Transitions of the PTR TTY clock continue at a 110.011 Hz rate until an entire 11 bit character has been shifted out to the printer, at which time an end of character gate becomes active and sets the data needed flip-flop which in turn resets the counter reset flip-flop. The printer control circuits are now back in their rest state and are ready to accept the next character from the processor.

### 3.3.3 Keyboard Control Circuits

When the keyboard control circuits are in their rest state, the data available flip-flop, the space flip-flop and the counter reset flip-flop are all in the cleared state. When the operator depresses a key, the first bit of the character, which is always a space, causes the space flip-flop to be set. This enables the counter reset flip-flop to be set synchronously with the 9.9 KHz

timing signal. The divide by 90 counter begins generating the KBD TTY clock which center samples the keyboard bits into the 10 bit serial to parallel converter.

Whenever the counter reset flip-flop is set, a transition detector circuit presets the serial to parallel converter to all marks, which enables the end of a character to be sensed by simply looking for a space in the last stage and a mark in the first state of the serial to parallel converter. When this occurs the output of an end of character gate becomes active and resets the space flip-flop which then causes the keyboard circuits to revert to the rest state. The output of this gate also sets a data available flip-flop which indicates to the processor that 8 keyboard data bits can be transferred using a CID instruction.

When the CID instruction is executed, the processor generates a data received signal (CH3DRL) which resets the data available flip-flop, causing the data available signal to go to the OFF state. The keyboard circuits can now accept the next character.

## SECTION 4

### ADM DESIGN STUDY REPORT

In this section we will describe a low-cost, high-speed, programmable processor which was conceived and is being developed at Philco-Ford and which is the recommended approach for implementations of the LONGBRAKE advanced development models. We call this machine the Programmable Array Processor (PAP). In the following paragraphs we will attempt to show how this novel processor architecture will provide the programmability of a general purpose computer, the throughput and simplicity of a pipeline processor, and the low cost and reliability required for operational field equipment.

#### 4.1 Consideration of Alternative Approaches

In addition to the programmable array processor which we are about to describe, a number of alternative approaches have been considered. In this section we will comment briefly on the advantages and disadvantages of the following four alternatives.

- I. Parallel LSI Microprocessors
- II. MSI/LSI Programmable Processors
- III. Minicomputer with LSI Peripherals
- IV. Custom Pipeline Processor

##### 4.1.1 Parallel LSI Microprocessors

The first approach, that of using a number of microprocessors in parallel, appears to be impractical because of the large number of microprocessors required to approach the lower bound on the

arithmetic throughput required in the LONGBRAKE system.

The LONGBRAKE algorithm requires about 4000 multiplications and additions per frame; about half in the transmitter and about half in the receiver. The frame rate is 44.4 frames per second and precision is 16 bits. This gives a computational rate of 177, 000 multiply and add combinations per second required for the arithmetic-and-logic-unit.

As an example of how microprocessors would handle this throughput, consider the presently available Intel 8008's which do an algorithm multiply for 8 bit precision. Running the algorithm for signed integers takes between 1000 and 1500  $\mu$ -seconds.

Using an 8 bit machine to do 16 bit multiplication would increase the running time by a factor of at least 4. This means that 16 bit multiplication using a single Intel 8008 would take between 4000 and 6000  $\mu$ -seconds giving a throughput of 167 to 250 multiplications per second. This means that about 1000 Intel 8008's would be needed to do nothing but the multiplications in the LONGBRAKE algorithm.

Of course it is recognized that faster microprocessors are being developed and may soon be available, but this example indicates that they must be several hundred times faster than the Intel 8008's before they will begin to be applicable to this system. In the long run, however, increasing the speed of these devices may be more than simply a matter of using faster technology, for at present the circuit speeds associated with the technology used in the fabrication of microprocessors is fairly high. The reduced throughput comes about due to the inherent pin limi-

tations when the entire processor is put on one chip. Most microprocessors use a single bidirectional data-bus for getting operands to and from the chip. This data -bus, forms a constriction to operand flow and results in a low efficiency for the arithmetic-and-logic circuits.

In addition to the low efficiency of microprocessors, the problems involved in programming a parallel system of arithmetic-and-logic units was considered formidable. For these reasons, parallel micro-processors are not recommended for the ADM implementation.

#### 4.1.2 MSI/LSI Programmable Processors

The determining factor in the evaluation of this approach is the concept of "programmable". An important distinction must be considered here. The term programmable processor is quite often used to describe a general-purpose machine which is designed to do a large class of computations. The general purpose machine may not be particularly effective for a smaller class of computations within the class for which it was designed. The present voice processor involves mainly on-line computations with little program modification. It is necessary in this case to consider not the fact that the machine is programmable, but rather that the machine is general-purpose. The result of this is that the duty cycle of the arithmetic-and-logic circuits for the general-purpose machine is rather low for on-line computations.

An example of the low efficiency of a general purpose machine for on-line computations may be seen by examing a section of coding for an arithmetic operation on two indexed operands over a range of the index. The majority of the machine time is spent in fetch-

ing instructions, fetching operands, doing address arithmetic, incrementing index registers and testing for the end of the index range; and very little of the time is spend doing the specified arithmetic on the operands. Because of this low arithmetic-and-logic efficiency, the general-purpose machine organization can provide the kind of throughput required for speech processing only by employing extremely fast logic circuits in the arithmetic unit and the memory, and by employing a large and powerful instructions set.

In practice very fast logic is expensive, consumes high power, and is difficult to implement with MSI and LSI. It also requires very careful layout, debugging, and maintenance to cope with timing problems due to excessive wire lengths and marginal components. Large instruction sets furthermore tend to increase system complexity and hence system expense, in terms of design, fabrication, and maintenance. For these reasons we do not recommend this approach for the LONGBRAKE ADM's.

#### 4.1.3 Pipeline Processors

Approaches III and IV are considered to be similar, assuming that the LSI peripherals of IV are organized as pipelines. It is well known that the pipeline organization provides both a high efficiency for the combinational logic circuits and a simple control section. However, the "custom" pipeline processor approach implied in III and IV as defined here would probably require more than a single combinational logic unit. For a low cost approach, the required throughput of the LONGBRAKE algorithms can be easily achieved by a single arithmetic-and-logic unit.

#### 4.1.4 The Recommended Approach - A Programmable Array Processor

The approach which is recommended for implementation of the LONGBRAKE ADM's is similar to approach IV in the organization of memory and processing section but uses a single arithmetic-and-logic unit. Furthermore, it is recommended that the control section incorporate a ROM for storing the program and that the instruction set be chosen to make the machine effective for the type of computations involved in the specific algorithms under consideration and not for general-purpose computations.

The end result of these considerations was the selection of a programmable array processor which is described in Section 4.2. This processor is believed to have the gate efficiency of the pipeline processor (Approach IV) while still maintaining much of the programmable features of Approach III. The instruction set has been selected to give high efficiency for indexed array arithmetic at the expense of general computational capability. One might say that, although the machine retains the concept of programmability, it has been tailored for high efficiency in specific voice processing algorithms. As a result the programmable array processor can easily achieve speech processing throughputs greater than those of the fastest currently available general purpose processors.

In addition to the extremely high gate efficiency for the voice processing algorithms, there were several other advantages of the programmable array processor that were considered. First, the control section was relatively simple compared to the general-purpose machine. Consequently, a high proportion of the total

number of modules involved in the implementation would be in the memory. Here arithmetic throughput is not a major factor in selecting circuit types and LSI, and particularly C-MOS LSI circuits, may be used to their best advantage. In fact, there are several presently available "off-the-shelf" LSI chips which can be used for the array memory.

A second advantage in the use of the programmable array processor is that truncation, shortening, or otherwise modifying computations to reduce running time is not necessary. In fact, the programmable array processor functions best when a given indexed array computation is done exactly and completely. This should provide improved speech quality over present methods of implementation.

A final advantage is that the machine accomplishes high-speed processing while using memory and logic circuits which are for the most part relatively slow. As a result the problems and expense of layout, debugging and maintenance of very-high-speed circuits are minimized.

## 4.2 Description of Recommended Hardware

### 4.2.1 Programmable Array processor Concept

The Programmable Array Processor is designed to be extremely efficient for computations involving on-line, indexed, arithmetic or logic operations. These are computations where the operation and one or more of the operands are specified over a given index range. An elementary example is the computation of the one dimensional array of products,  $p_i$  where:

$$p_i = a_i b, (i = 1, 2, 3 \dots I)$$

Actually the processor is designed to most efficiently handle double indexed operations, such as the computation of the two dimensional array of products,  $p_{ij}$ , where:

$$p_{ij} = a_i b_j, (i = 1, 2, 3 \dots I) (j = 1, 2, 3, \dots J). (4-2)$$

The instruction set is designed so that each of the above arrays can be computed and loaded into memory with a signal instruction. For programs involving indexed arithmetic operations, the amount of program memory required is extremely small compared to that for the more general-purpose machine.

A further example of the effective use of this processor is the computation of an array of correlation coefficients,  $A_k$ , where:

$$A_k = \sum_{i=1}^N a_i a_{(i-k)} \quad (4-3)$$

The operation of multiplication and accumulation over an index can be accomplished with a single instruction so that one instruction would generate the coefficient  $A_k$  of Equation (4-3).

The Programmable Array Processor configuration as presented in this report is intended to run real-time programs and to operate as communication terminal equipment rather than as a laboratory tool. Consequently, the design departs significantly from that of the conventional general-purpose computer. The first departure is the use of separate operand and program memories, as opposed to the more conventional use of a single memory for both operands and instructions. Separating the memories precludes tradeoffs between program and operand storage requirements. This is not a disadvantage in the intended application, since the required storage

for both program and operands is fixed. Also, the use of separate memories precludes the modification of the program during execution. Again this is not a disadvantage for communication equipment since the program must be stored in non-volatile memory and should not be modifiable under any ordinary conditions.

The second departure from the conventional computer design is the use of a triple address instruction and the capability for three simultaneous memory accesses. This feature greatly increases the duty cycle of the arithmetic unit and removes the constriction of data flow imposed by a single memory bus.

Although the concept of a triple address machine is not new, the concept is usually not implemented because of the inherent multiplexing and demultiplexing problems in a triple access memory. Most conventional random-access memories are designed, for economic reasons, to multiplex all memory locations onto a single bidirectional bus. Furthermore, reading and writing of conventional random-access memories cannot occur simultaneously.

This single memory bus, inherent in most random-access machines, may be viewed as a constriction in operand flow to the arithmetic unit. In general, three operands must be passed back and forth by the bus for each combining operation. The result is that the arithmetic unit operates on a relatively low duty cycle.

This sequential memory access problem is well known. Measures are usually taken in the design of high-speed machines to reduce the time lost in fetching operands. A common means of reducing memory fetches is to provide a temporary register file which acts as a second source of operands to the arithmetic unit. The

array memory, with triple simultaneous access, may be viewed as a further step towards making operands accessible to the arithmetic unit and towards removing the constriction of a single memory bus.

FIGURES 4-1 and 4-2, in which heavy lines indicate memory data paths, illustrate the differences in the arrangement of memory and arithmetic units between the conventional processor and the Programmable Array Processor. FIGURE 4-1 shows the conventional arrangement using a single random-access memory and FIGURE 4-2 shows the organization for the Programmable Array Processor. In FIGURE 4-1 only one memory address can be applied to the memory at any time and only one operand can be read from or written into memory. For each combining operation, three memory accesses must be performed sequentially. In FIGURE 4-2 the three memory addresses are supplied simultaneously to the operand memory. Both source operands immediately become available to the arithmetic-and-logic unit; and the combination is loaded into the destination as soon as the arithmetic operation is completed. Note that the constriction to operand flow, imposed by a single memory data register, has been removed. For indexed arithmetic or logic operations, one memory operation occurs for one combining operation. Furthermore, the machine of FIGURE 4-2 can be designed so that instruction fetches overlap operand accessing for indexed computations.

The third departure from the conventional processor is the use of the array memory. The array memory contains a number of circulating arrays each of which is addressable and each of which is multiplexed onto two memory source buses feeding the arithmetic unit. A third bus, the destination bus, is demultiplexed to each

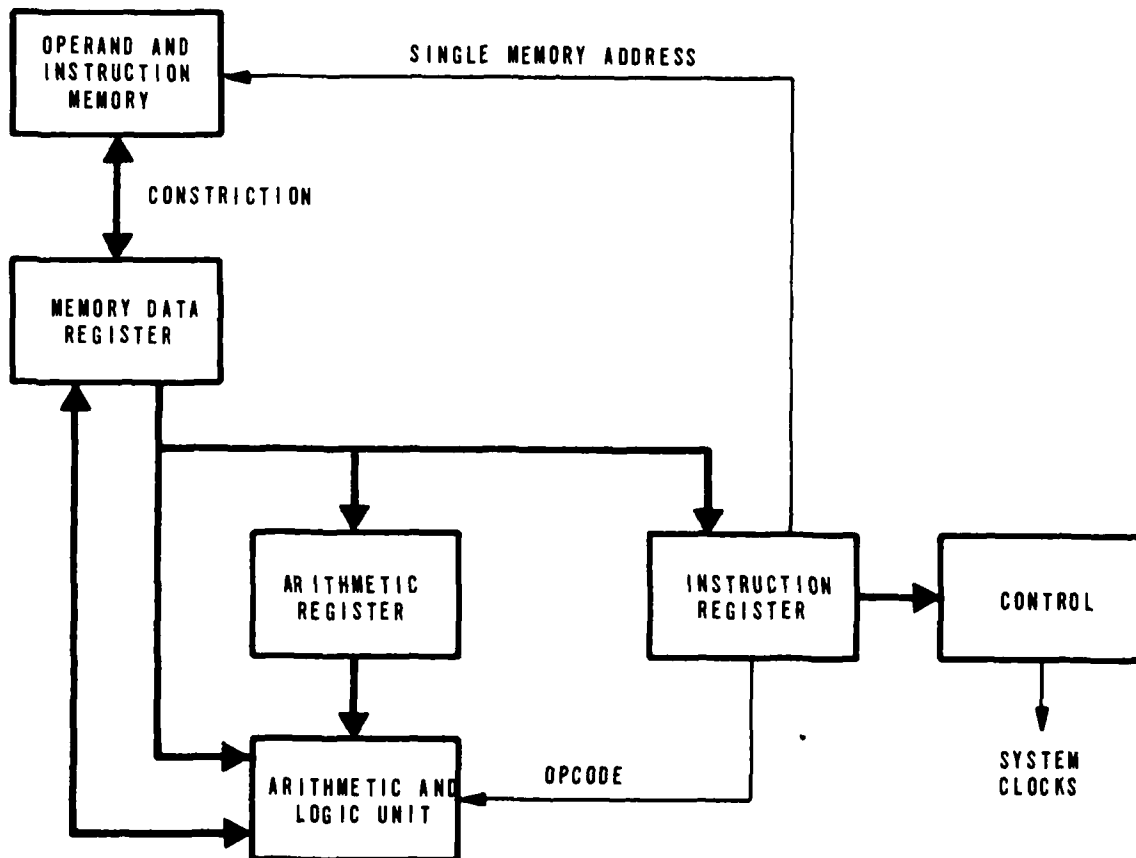


Figure 4-1 - Conventional Processor Organization

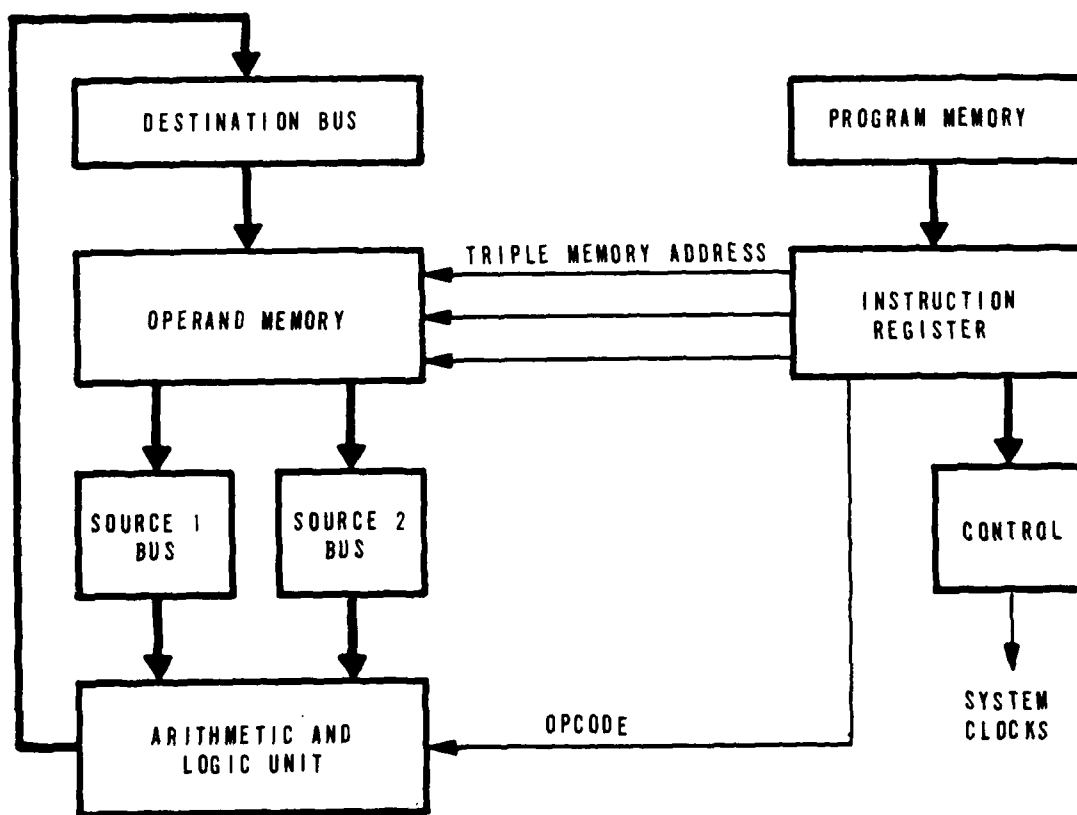


Figure 4-2 - Programmable Array Processor Organization

array. Since the number of arrays is only a small fraction of the number of operands stored, the number of multiplexing and demultiplexing circuits does not become prohibitive. Furthermore, since the input to an array is physically a different circuit from the output, reading and writing can occur simultaneously. Three arrays may be addressed simultaneously and any array may act as: the first operand to be combined; the second operand to be combined; or, the destination for the combination. In fact, an array can act as both source and destination.

Since the arrays are circulating rather than random-access, any array may be viewed as a sequential memory. In signal processing applications this fact often becomes advantageous. If the programmer loads the arrays in the order in which operands are to be combined, address arithmetic is eliminated. Incrementing an address is equivalent to shifting an array. Each time an array is loaded, the address may be considered incremented.

Shifting of operands in one array relative to those in a second is accomplished by simply applying one or more shift pulses to the first array. This feature greatly simplifies generation of correlation coefficients.

As described in Section 4.2.2.1, one memory location is random-access rather than circulating. This permits transfer of arrays from circulating memory to random-access memory. By moving selected arrays to random access memory, operands can be reordered or any other operation requiring random-access can be performed.

#### 4.2.2 Programmable Array Processor Implementation

The paragraphs to follow describe a PAP device that can be

built using off the shelf FIFO's and which will operate at a 1 MHz clock rate. This rate provides more than enough computational power to handle the LONGBRAKE algorithms. However, it should be stated that there is no fundamental reason why the clock rate could not be raised to 2 MHz if 2 MHz shift register memory becomes available. Custom LSI shift register memories and the remainder of the logic could indeed operate at the 2 MHz rate with very little modification.

The hardware for the Programmable Array Processor is divided into three major assemblies which are treated separately in the following sections of this report. The three major assemblies are:

① The operand memory; ② The program memory and control section; and ③ The arithmetic-and-logic unit. Overall operation of the processor is described using a timing diagram to illustrate the execution of each class of instruction which the processor is intended to run. The timing diagrams for the instruction execution define the permissible delay through the various combinational logic networks in the program memory and control sections, and also gives the execution time for each class of instruction in terms of the basic machine cycle. A state-diagram illustrates the total number of machine states required and the amount of storage needed in the control section.

The hardware is designed for an arithmetic throughput of 1 mHz, at 16-bit precision. The basic machine cycle time is 1 usec. The machine as configured may also be implemented without custom circuits.

#### → 4.2.2.1 Operand Memory

The operand memory, shown in FIGURE 4-3 consists of: 16 addressable memory locations, labeled 0 through 15; multiplexing circuits for multiplexing each memory location into the arithmetic-and-logic unit; and, a destination bus for demultiplexing the output of the arithmetic-and-logic unit into memory. Each memory location can contain a sequential-access array, a random-access array, and accumulator, or a single operand. Each random-access array inherently occupies two addressable locations, one for the address and one for the operands.

In order to be efficient, with respect to the number of memory addresses used, the operand memory should consist primarily of sequential access arrays.

In FIGURE 4-3 memory locations 4 through 15 are shown as sequential access arrays implemented with FIFO's\*. The address of the operand read from a FIFO is not specified explicitly since the operand appearing at the FIFO output is a function of the number of dump clocks applied to the FIFO. Addressing of the entries within a FIFO array are under program control in the sense that the instruction specifies whether or not a dump clock is applied after an operand is read.

In the case of random access arrays the address must be explicit. Each random-access array has an associated address register. This address register may be loaded or incremented by a constant via the program. Automatic incrementing by ONE is also provided by making the address register also act as a binary counter.

\*FIRST-IN, FIRST-OUT MEMORIES

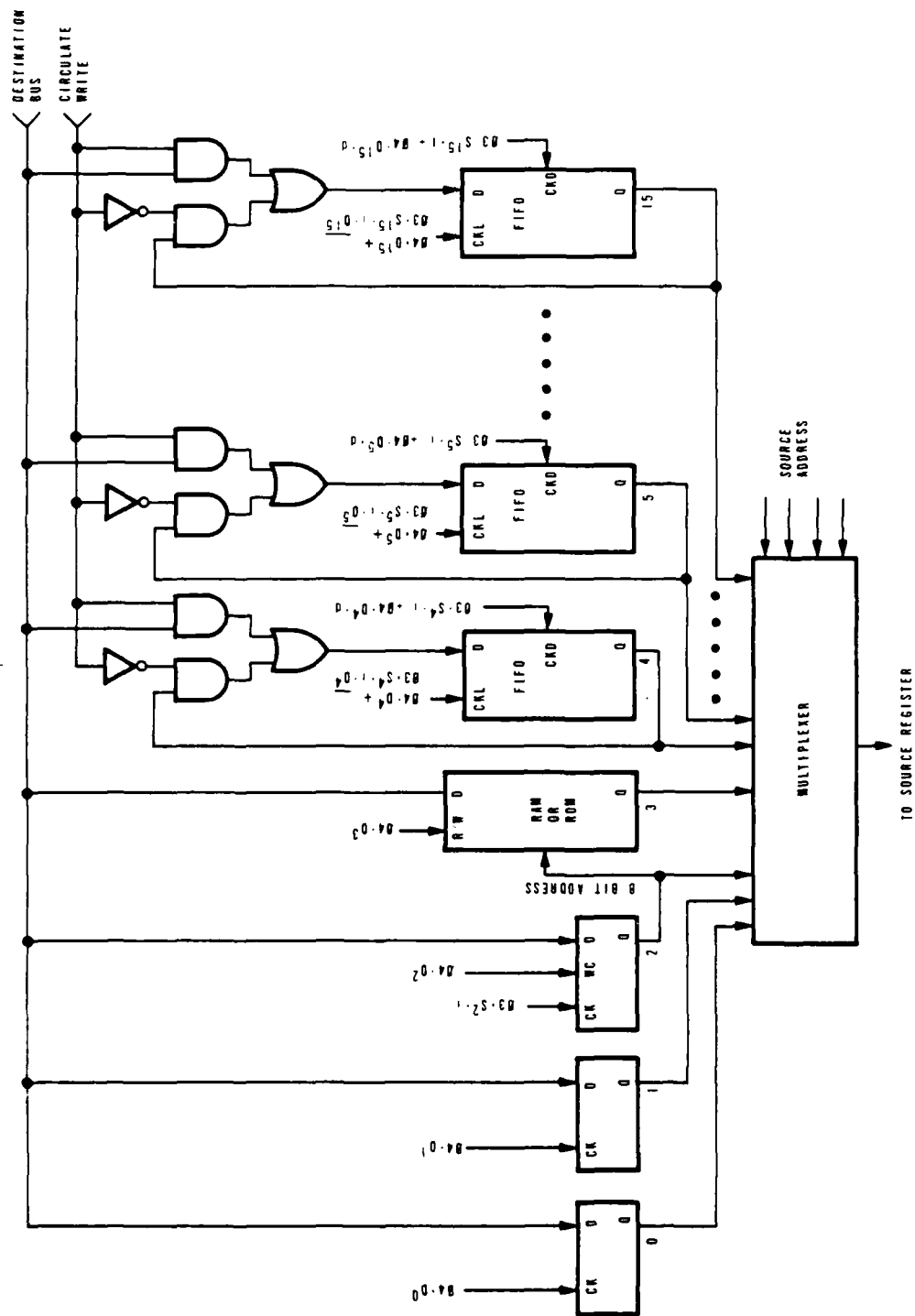


Figure 4-3. Operand Memory Block Diagram

In FIGURE 4-3 memory locations 2 and 3 illustrate the implementation of a random-access array.

Memory location 0 and 1 are single operand locations implemented with storage registers. These locations are used like the arithmetic registers in a conventional machine. Using locations 0 and 1 for temporary storage, arithmetic operations can be performed where the operation involves three locations in one random access array.

Any single-operand addressable memory location, such as locations 0 and 1 of FIGURE 4-3 may be replaced by an accumulator register. When the accumulator register is used as the destination of an indexed arithmetic operation the results of that operation are accumulated over the index range. This saves running time by permitting the accumulation of a number of products to be accomplished with an indexed multiply instruction using the accumulator register as the destination.

In addition to the sixteen addressable memory locations, consisting of the array memories and the single registers, the operand memory contains multiplexing circuits for supplying the operands from the addressed arrays to the arithmetic unit. The multiplexing circuits are implemented with sixteen  $T^2L$  circuits (TI number "150" 16-line to 1-line multiplexers). The memory address is decoded on the chip so that only the 4-bit binary source address is supplied to the multiplexer.

Since presently available FIFO's can only drive one  $T^2L$  load, and in order to minimize the number of chips required for multiplexing, one multiplexer is time-shared between the two source registers of the arithmetic unit. This is done by first loading

source register-2. The two clock-pulses for loading the source registers are shown in the second line of FIGURE 4-4 which contains the timing diagrams for memory operation.

Operands are supplied to the operand memory from the destination bus and must be loaded into the addressable memory location selected by the program. This demultiplexing function is accomplished by simply gating the clock pulse ( $\phi 4$  of FIGURE 4-4) to the selected destination. Typical equations for the clock decodes are given in FIGURE 4-3.

In the case of the FIFO's, the array is sequenced by dumping operands. If the FIFO were sequenced and not used as a destination for new operands its content would go to zero. Consequently, provision is made for re-entering operands into the FIFO as they are dumped, so that the array is not destroyed by reading. Each FIFO has a selection gate associated with its data input so that it may be loaded from the destination bus or from its own output. When a FIFO is selected as a source, and operands are dumped, and that FIFO is not a destination for arithmetic results; then those operands dumped will be re-entered via the selection gates. The selection gates for each FIFO require 4 TI type "157" T<sup>2</sup>L chips.

The timing of the memory operations is illustrated in FIGURE 4-4. Four basic system clocks are used,  $\phi 1$  through  $\phi 4$ . The first phase of a sequence,  $\phi 1$ , loads the instruction register and makes the first source address available to the memory multiplexer. The first pulse of  $\phi 2$  loads the source-2 register of the arithmetic unit and simultaneously switches the source-1 and source-2 addresses in the instruction register. The second pulse

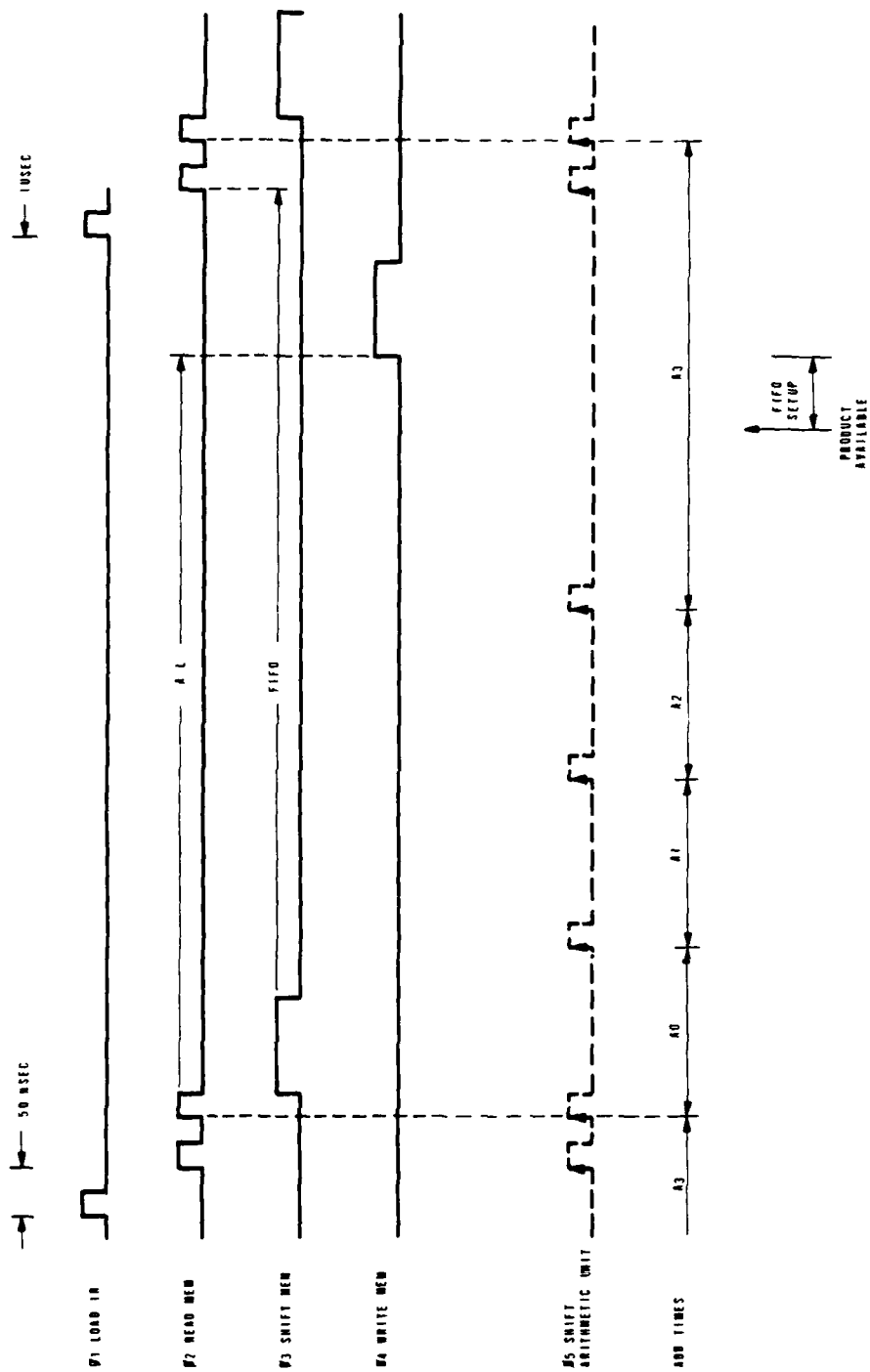


Figure 4-4. Memory Operation Timing Diagram

of Ø2 transfers the contents of the source-2 register to the source-1 register in the arithmetic unit, and loads the source-2 register from memory. Both the source-1 and source-2 registers of the arithmetic unit are thus loaded by Ø2.

The fourth phase Ø4 loads the addressed memory destination. Following Ø4, the memory cycle repeats.

The propagation time allowed for the arithmetic-and-logic unit is 750 nsec, and that for the FIFO is 825 nsec.

Although the four clock phases of FIGURE 4-4 have been defined in terms of their operand memory functions, these same clock phases are used to operate the program memory and control section and are shown in FIGURE 4-6 on a smaller scale as the basic clock sequence for execution of the five major instruction types. The fifth clock phase of FIGURE 4-4 is used only by the arithmetic unit.

#### 4.2.2.2 Program Memory and Control

The program memory and control section provides the means for storing, fetching, and executing instructions. The execution of instructions is the most complex operation and will be described first using the instruction timing state diagram of FIGURE 4-5.

In order to count basic machine cycles, a 2-bit instruction timing register, referred to as IT, is used. The instruction timing register has 4 states (IT1, IT2, IT3, and IT4) shown as circles in the state diagram of FIGURE 4-5. Within each circle is listed the primary operation which may occur in that state. Below each circle is listed the primary operation with which the state is associated. IT1 is used primarily for execution of non-indexed

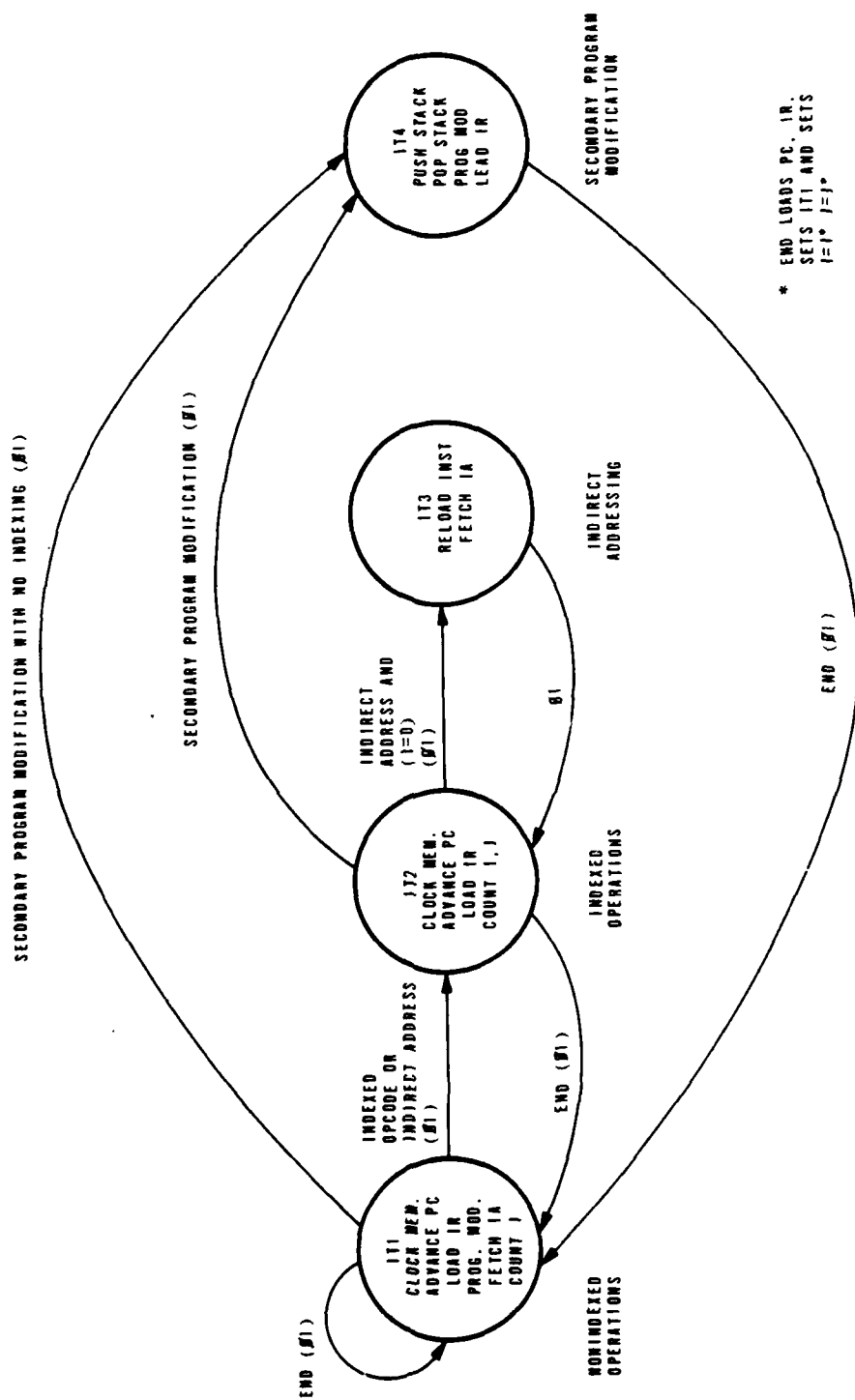


Figure 4-5. Instruction Timing Diagram



instructions. IT2 is used primarily for indexed operations. IT3 is used for fetching indirect addresses, and, IT4 is used for instructions for program modification following an arithmetic operation.

In the case of instructions involving only one machine cycle, many of the functions of IT2, IT3, and IT4 may be done in IT1.

The IT register always changes state on  $\phi 1$  (FIGURE 4-4). The next IT state is determined by the OPCODE, OPX, and PGX standing in the instruction register, and the content of the I, J, counters. A secondary decode END, derived from the above information and indicating the end of an execution, is shown in FIGURE 4-5. The decodes indicating the next state to which IT goes are indicated by the arrows.

The instructions for the array processor have been divided into five types based on the sequence of IT states involved in their execution. These five types and their timing sequence for execution are illustrated in FIGURE 4-6. A typical example of each type of instruction is also given. A new instruction is loaded into the instruction register on the generation of an END signal and a  $\phi 1$  clock. The END signal also sets IT to IT1 on  $\phi 1$ . Therefore, each instruction execution begins on  $\phi 1$  in IT1.

The first instruction type is memory operation with operand indexing only. A typical example is the ADD instruction with indexing on one source and the destination. This type of instruction is executed in one machine cycle in IT1. The  $\phi 2$  clock loads the arithmetic unit; the  $\phi 3$  clock increments the indexed source and advances the program counter; the  $\phi 4$  clock loads and increments

the destination; and, the next 01 clock reads the next instruction.

The second instruction type is the control instruction. A typical example is the SET instruction which simply specifies the range of I and J. This instruction does not involve operand memory. It is executed using 03 to load the I\*J\* registers from the instruction register and to increment the program counter.

The third type of instruction is memory operation with operation indexing. This type of instruction begins in IT1 where the operation specified by the OPCODE is performed once as with type 1. However, no END is generated in IT1; rather, the specified index register is decremented and IT2 is set. The operation continues in IT2 until the specified index register reaches 1. At this point an END is generated, IT1 is set, the index register is reset and a new instruction is loaded into the instruction register.

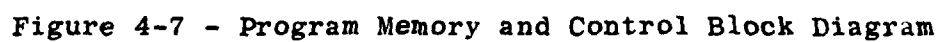
The fourth type of instruction is memory operation with secondary program modification. An example of this type is a single ADD without indexing but a PGX indication to loop if I  $\neq$  0. After performing the memory operation in IT1, the instruction timing is set to IT4. In IT4, the program counter is not incremented but rather loaded from the address stack and I is decremented. (It is assumed I was not ZERO.) In IT4, an END is generated, and the next IT state is IT1. Had I been ZERO in IT4, the program counter would have been incremented and the most recent entry in the address stack would have been deleted.

The fifth type of instruction involves indirect addressing. This implies a double index on the OPCODE and at least one operand.

In this case, the operation is performed in state IT1 and IT2 much like the type-1 instruction. However, the result of the memory fetch in IT1 is not used as an operand but is loaded into the instruction register as an indirect address. At the same time, the source of indirect addressing is dumped and J is decremented. In IT2, the specified operation will be performed until I goes to ZERO. At this time, IT3 will be set, the instruction register will be reloaded and the pre-indexed indirect address will again be loaded into the instruction register. The instruction timing will again be set to IT2 and the operation performed until I again goes to ZERO at which time IT3 will again be set. When in IT2 and both I and J go to ZERO, an END is generated, the program counter is incremented, IT1 is set, I and J are reset, and a new instruction is begun.

FIGURE 4-7 is a block diagram of the program memory and control section. This section, exclusive of the ROM's which store the program and the control decodes, is implemented with T<sup>2</sup>L. The registers are "D-Type" edge triggered registers with a load input. The load input is used in place of clock gating in order to avoid clock skew.

Two clock phases are used in the control section;  $\phi 1$  and  $\phi 3$ . The  $\phi 1$  clock sets the instruction register, IR, the instruction timing register, IT, the i-index register I, and the j-index register J. Decodes from these registers are used to generate the load signals for the registers themselves as well as the data for the IT registers, the load and data select signals for the program counter PC and the address stack ST, and the load



signal for the I\*J\* registers which hold the specified ranges of i and j.

The Ø3 clock is used to control the program counter PC, the I\* and J\* registers, and the address stack ST. The address stack may be thought of as a LIFO<sup>1</sup> memory which is loaded from the program counter and which stores the starting addresses of loops marked by the program.

#### → 4.2.2.3 Arithmetic-and-Logic Unit

The arithmetic-and-logic operations required to execute the Programmable Array Processor (PAP) instructions are performed on an arithmetic and logic unit as shown in block diagram form in FIGURE 4-8. Operands are first read from the operand memory applied to the arithmetic and logic unit, and the result written back into the operand memory. The operations which the arithmetic and logic unit is capable of performing are addition, subtraction, multiplication, right shifts of from one to fifteen bits, left shifts of from one to fifteen bits, and table lookup.

Since the arithmetic and logic unit generates a result in 750 nanoseconds, which is less than the operand memory cycle time (1 microsecond), it does not add any additional delay to just recirculating a FIFO memory upon itself. This simplifies the writing of results into the specified locations in the operand memory because the instruction register still contains the required write address.

To minimize the amount of logic circuits required, the arithmetic and logic unit was designed to perform its computations in 4 steps; i.e., the 750 nanosecond total time is divided into 4

<sup>1</sup>Last-in, first out memory.

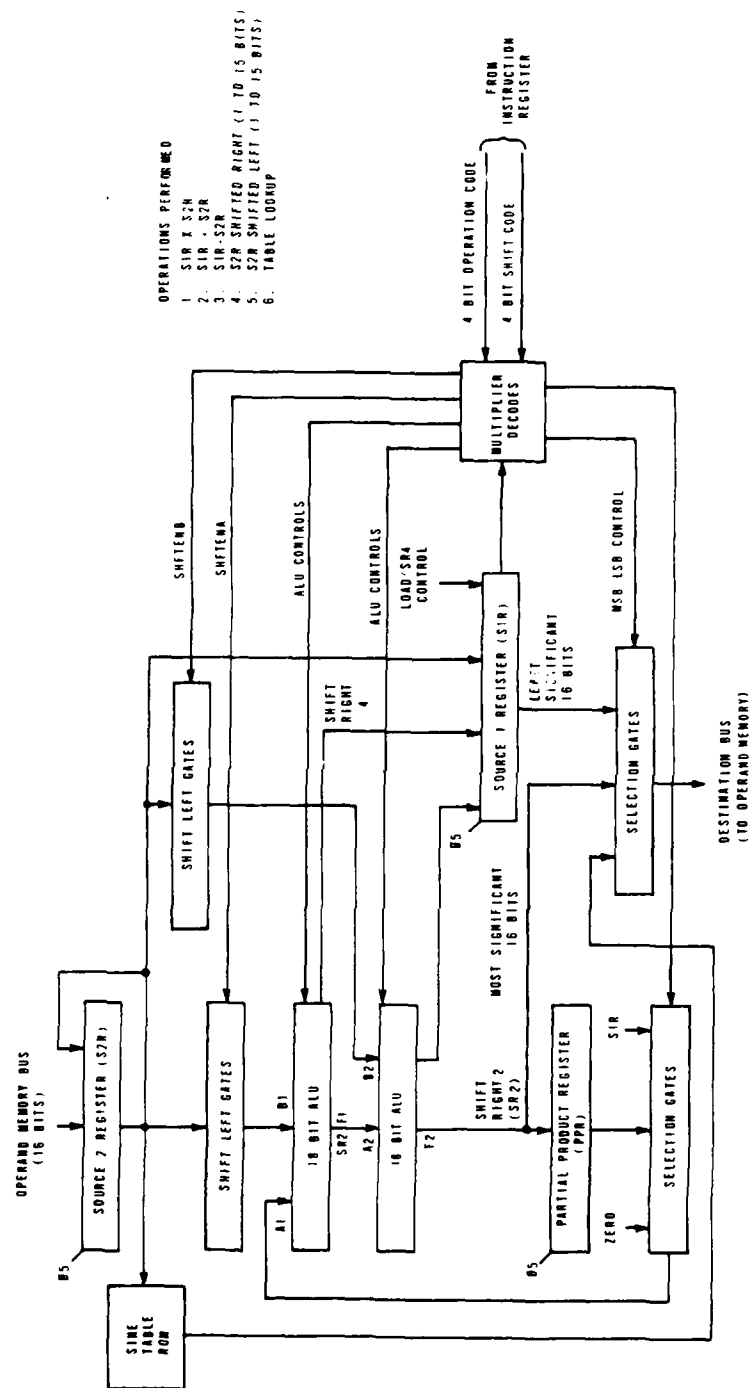


Figure 4-8. Arithmetic and Logic Unit Block Diagram

equal periods and the same arithmetic circuits are used during each period. This corresponds to a clock rate of approximately 5.4 mHz required for the arithmetic-and-logic unit. An 8-step process was also considered, but was discarded because it was found the excessively high clock rate resulted in propagation delay problems.

All registers in the arithmetic-and-logic unit are clocked by  $\phi 5$ , which is shown in FIGURE 4-4. Also shown in this figure are the times corresponding to the four steps of the arithmetic process, called add times, and labeled A0, A1, A2, and A3.

During A3, the source-2 register (S2R), shown at the upper left in the block diagram of FIGURE 4-8 does not recirculate upon itself as it does during the other three add times but is connected to the operand memory bus, which is the output from the operand memory. The clock pulse occurring near the end of A3 clocks the first operand into S2R. The clock pulse occurring at the beginning of A0 transfers the contents of S2R into the source-1 register (S1R) and simultaneously loads the second operand from the operand bus into S2R. We now have both source registers loaded with operands from memory. During A0, A1, and A2, S2R recirculates upon itself and S1R shifts to the right 4 bits each time a clock pulse occurs. If a multiply problem is being performed then S1R holds the multipliers and S2R holds the multiplicand. The following paragraphs describe what occurs during the four steps of the arithmetic process (A0, A1, A2 and A3) for each of the possible operations performed.

The contents of register S2R are applied to the B1 input

of an 18-bit arithmetic logic unit (ALU) either directly or shifted left one bit. Shifting left is performed by the set of 16 two input selection gates shown in the block diagram of FIGURE 4-8. The other input to the ALU, labeled A1, comes from another set of selection gates whose output can be ZERO, the partial product register (PPR), or the source-1 register (SlR). The F1 output of the ALU can be any of the following three functions of A1 and B1 depending on the ALU controls:

$$F1 = A1 + B1$$

$$F1 = A1 - B1$$

$$F1 = A1$$

The F1 output of the first ALU is connected to the A2 input of a second ALU with a wired in right shift of 2 bits. The B2 input to this ALU is another set of shift left gates connected to S2R. The F2 output is fed to a partial product register with a wired in right shift of 2 bits. Therefore, if both ALU's perform the function  $F = A$ , then each time the PPR is clocked it will just shift to the right by 4 bits.

Multiplication is performed by decoding the two least significant bits (the  $2^0$  and  $2^1$  bits) of SlR into control signals for the first ALU and into a shift enable (SHFTENA) signal for the shift gates which feed its B1 input. The  $2^2$  and  $2^3$  bits of SlR are likewise decoded into signals which control the second ALU and the shift gates which feed its B2 input. All of this decoding is done by the circuits labeled multiplier decodes in the block diagram.

The multiplication process starts in A0, at the beginning of which both operands have been loaded, and continues until A3. Since

S1R shifts to the right by 4 bits during each add time, by the time A3 is reached all 16 bits of the multiplier have been used; i.e., have been applied to the multiplier decodes and used to control addition or subtraction of the multiplicand (stored in S2R) to the partial product.

As S1R shifts to the right, the least significant bits of the product from the ALU's shift into the left so that when A3 is reached, S1R contains the 12 least significant bits of the product. Toward the end of A3, a full 32 bit product is available, but since the operand memory stores 16 bit words, a set of selection gates is provided to choose the most significant or least significant 16 bits. If a fractional multiplication is being performed then the most significant 16 bits are chosen; for integer multiplication, the least significant 16 bits are chosen under control of the multiplier decodes.

The multiplier decodes receive a 4-bit operation code from the instruction register which indicates which operation to perform. If the operation is shifting, then a 4-bit shift code controls the number of shifts in a manner to be described in the following paragraphs.

To perform addition the 4-bit operation code is decoded and control signals generated which route the contents of S1R through the selection gates shown below PPR to the A1 input of the first ALU, and route the contents of S2R to the B1 input. The first ALU is set to perform the function  $F1 = A1 + B1$  and the second performs  $F2 = A2$  during A0. During A1, A2, and A3, both ALU's perform the function  $F = A$ , so that the sum generated during A0

just circulates through the ALU's and the PPR with a shift right of 4 bits each add time. The selection gates which route data onto the destination bus are set to select the 16 least significant bits, which for this case are not a product but a sum.

Subtraction is performed identically to addition, except that  $F1 = A1 - B1$  is done during add time A0.

Right shifts are performed using the two sets of shift left gates and making use of the wired in shift right of 4 when PPR recirculates through the ALU'S. To obtain a right shift of one bit, during add time A0, the A1 input to the first ALU, is clamped to zero and  $F = A$  is performed by both ALU's. During the A2 and A3,  $F = A$  is repeated by both ALU's. In effect, we are recirculating all ZEROS. Then during add time A3, SHFTENB is activated, shifting the contents of S2R one bit left before applying it to the B2 input of the second ALU (its A2 input is ZERO at this time). Since there is a wired in shift right of 2 between the output of the second ALU and PPR, the net result is a shift right of one bit.

A right shift of 2 is obtained by repeating the process described above for a right shift of 1, except not activating SHFTENB during A3.

For a right shift of 3, SHFTENA is activated during A3, shifting S2R one bit left and applying it to the B1 input of the first ALU (the A1 input is ZERO). The second ALU performs  $F2 = A2$  during add time A3.

For right shift of 4, the process is identical to that for a shift of 3, except that SHFTENA is not activated.

Shifts of 5, 6, 7 and 8 are performed by always doing a constant right shift of 4 during A3 (by just recirculating PPR through the ALU's) and performing right shifts of 1, 2, 3, or 4 during A2, depending upon whether total shifts right of 5, 6, 7, or 8 are desired, respectively.

Shifts of 9, 10, 11 or 12 are performed by performing right shifts of 4 during both A2 and A3 and right shifts of 1, 2, 3, or 4 during A1, depending on whether a total shift of 9, 10, 11, or 12 is desired, respectively.

Left shifts are performed by first subtracting the number of left shifts from 16, then performing this number of right shifts and finally transferring the 16 least significant bits to the operand memory (which is identical to shifting left by 16 before performing the transfer).

Table lookup is performed using a read only memory (ROM) whose address input is taken from S2R. Whenever the 4-bit operation code indicates table lookup, the selection gate which drives the destination bus routes the output of the ROM onto this bus.

#### 4.3 Programming the Processor

The Programmable Array Processor (PAP) is tailored primarily for efficient processing of arrays of numerical data, since this constitutes the bulk of the operations in most signal processing applications. Initially, it was felt that this type of processor, being designed specifically for signal processing applications, would not be as powerful and flexible as conventional programmable processor for other types of processing. Further

investigation, however, demonstrated that it is possible to configure the PAP so that the same features which provide array processing power, also provide considerable flexibility, speed, and program efficiency for conventional programs.

From a programming point of view, the advantages of this machine can be summarized as follows:

- . Entire arrays can be processed by as few as one instruction.
- . The 3-address, 3-bus structure of the PAP makes each instruction equivalent to 3 instructions on a conventional computer.
- . The "Source:Source:Destination" concept provides complete memory reference flexibility with a small program set.
- . Several unique and powerful signal processing instructions are available in the instruction set.
- . Loops are fast and simple to program: because indexing, testing, and jumping are automatic.

In the remainder of Section 4.3.1 we will describe these advantages of the PAP in greater detail and will provide some general examples to illustrate their significance.

#### 4.3.1 Array Processing

It is easily seen how processing entire arrays with just a few basic instructions can simplify the programming task and can reduce the number of program steps in any algorithm to which it is applicable. This kind of array processing is most efficient and can be used whenever the following two conditions apply:

- a) All elements to be processed in a particular array are to be acted upon by the same operation or sequence of

operations.

- b) The elements of all involved arrays are to be processed in numerical sequence according to either an I or J index where the index increment has unity magnitude.

By including instructions such as MAC (multiply and accumulate), MACD (multiply and accumulate in double precision), and AMDF, (accumulate the magnitude of differences) in the instruction set, many of the time consuming processes of linear predictive coding, filtering, and spectral analysis can be accomplished with single instruction loops. When the array operation to be performed cannot be described by a single instruction, there is no change in the automatic nature of the array indexing system, but the processing time for each array element increases in proportion to the number of instructions required to perform each repetition of the process.

#### 4.3.2 Efficiency of 3-Address, 3-Bus Structure

The 3-Address, 3-Bus structure of the PAP makes most instructions equivalent to 3 instructions on a standard computer such as the CSP-30. This can be demonstrated by the following simple example in which the contents of two memory locations, Source One (S1) and Source Two (S2), are multiplied by each other and stored in a third location, Destination (D).

EXAMPLE:

PAP

MUL:S1\*S2 → D

CSP-30

MMR:S1 → A

MULMA:S2\*A → A

MRM: A → D

This example illustrates the reductions in program length and also shows one reason why the PAP is much faster than a

more conventional machine. To perform this operation, the PAP reads the instruction, reads the two source memory locations, performs the operation and writes the result all in one memory cycle time. The more conventional CSP-30 must perform 3 program memory reads, 2 data reads, and 1 data write in 6 consecutive memory reference operations; thus, making it a fundamentally slower machine.

#### 4.3.3 Memory Reference Flexibility Provided by Source-Source-Destination Concept

Each instruction on the PAP includes two source memory references and a destination memory reference. It is designed this way to accommodate arithmetic instructions, all of which involve two inputs and one output. By including the two input (Source) addresses and the output (Destination) address in the instruction, complete source-source-destination flexibility can be achieved with a very simple instruction set. The significance of this can be seen by comparing the PAP instruction set with that of the CSP-30 which is a relatively powerful conventional signal processing computer. Most of the arithmetic and logical instructions on the CSP-30 have 10 different variations to provide some source-destination flexibility. The comparison is made in the following example for the ADD instruction.

<u>PAP</u>	<u>CSP-30</u>
ADD (S1:S2:D)	ADD AA, ADD AM, ADD AR, ADD DA, ADD IA, ADD IR, ADD LA, ADD MA, ADD MM, ADD MR

The CSP-30 is a fast conventional computer partly because of the source-source-destination flexibility that its large

instruction set provides. Anyone experienced with programming the CSP-30 knows, however, that there are a number of useful variations that have been omitted from its source-source-destination repertoire.

The PAP, on the other hand, provides all possible source-source-destination combinations with a truly minimal instruction set. This has obvious advantages to the programmer who does not have to memorize the meanings of large number of confusing instruction mnemonics, and it also helps to explain why the control hardware required in the PAP is so much simpler than that of the CSP-30. In this regard, the PAP is more like the very simple and relatively inexpensive computers such as the Data General Nova computers. The NOVA instruction set is also minimal with only one version of each instruction. But the NOVA's provide almost no source-source-destination flexibility in the basic instruction since the sources are always two out of four possible accumulators and the destinations can only be one of the two chosen source accumulators. To get the data out of and back into memory requires two additional fetches and a store. This accounts, in part, for the relatively low speed of these inexpensive machines.

#### 4.3.4 Special Features of the PAP Instruction Set

The PAP instruction set is listed in Appendix A. In addition to all of the basic instructions that are normally included in signal processing computers, there are several unique instructions which have been included to increase the power of the machine for speech processing, digital filtering, and spectral analysis. These are the MAC, MACD, and AMDF instructions. MAC and MACD are multiply and accumulate with single and double word outputs

respectively. These instructions are included because multiplication, and accumulation is the basic operating mode in convolution, correlation and discrete Fourier transformation. These instructions account for more than half of the operations in most LPC speech processors and modems. By combining multiplication and accumulation in a single instruction, each repetition can be performed in the basic PAP instruction time of 1.0 microsecond (assuming a memory cycle speed of 1.MHZ).

By contrast, an indexed double word multiply and accumulate on the CSP-30 requires nearly 3.0 microseconds even with a memory cycle speed of 10 MHz. These instructions, and the automatic indexing feature provided by the PAP, are probably the most significant factors in increasing the throughput of this machine for signal processing applications.

The third special instruction is the AMDF instruction, which is the accumulation of the magnitudes of the differences of S1 and S2. This operation is the basis for pitch estimation in the system, and has to be repeated a very large number of times in the process of computing one pitch number. By implementing these operations in one instruction, a process which requires an average of 2.1 microseconds on the CSP-30 can be accomplished in 1.0 microseconds on the PAP.

#### 4.3.5 Programming Loops on the PAP

Loops are fast and simple to program on the PAP because indexing, testing, and jumping are automatic.

Figure 4-9 illustrates the significance of the instruction

OPCODE   OPX	ADDRS S1   X	ADDRS S2   X	ADDRS D   X	PGX	
$\begin{pmatrix} I \\ J \\ IJ \end{pmatrix}$	$\begin{pmatrix} i \\ j \end{pmatrix}$	$\begin{pmatrix} i \\ j \end{pmatrix}$	$\begin{pmatrix} i \\ j \end{pmatrix}$	$\begin{pmatrix} L \\ M \end{pmatrix} \begin{pmatrix} I \\ J \\ IJ \end{pmatrix}$	
<u>X</u>	<u>SIGNIFICANCE</u>				
00	-	Do Not Increment Address			
01	i	Increment Address			
10	j	Increment Address if i = 0			
11	ij	Indirect Address, Pre-indexing with j, post-indexing with i			
<u>PGX</u>	<u>SIGNIFICANCE</u>				
000	NOP No Operation				
001	LI Decrement I, Loop if I≠0. Otherwise pop stack and proceed.				
010	LJ Decrement J, Loop if J≠0. Otherwise pop stack and proceed.				
011	LIJ Decrement I, Decrement J if I=0, Loop if J≠0. Otherwise pop stack and proceed.				
100	M Push the Contents of PC into the Stack.				
101	MI Push the Contents of PC into the Stack and load I.				
110	MJ Push the Contents of PC into the Stack and load J.				
111	MIJ Push the Contents of PC into the Stack and load IJ.				
<u>OPX</u>	<u>SIGNIFICANCE</u>				
00	S	Run the Instruction once.			
01	I	Run the Instruction I times.			
10	J	Run the Instruction J times.			
11	IJ	Run the Instruction IJ times.			

Figure 4-9. Significance of Instruction Fields

fields. The instruction format itself is shown at the very top of the figure.

The first field, OPCODE, specifies the operation to be performed such as ADD, MULTIPLY, ETC. The OPX field specifies the number of times the operation is to be performed. This can be either I times, J. times, or IJ times. The actual values of I and J are specified by the SET instruction and remain in effect during execution until respecified in the program.

Each of the three address fields has an index field, X, which can be either I, J, or IJ. An I index means shift the array each time the instruction is run, a J index on an address means shift the array each time the instruction is run and the I counter is ZERO. An IJ index on an address specifies indirect addressing with preindexing by J and post-indexing by I.

The program indexing field, PGX, is used for controlling multi-instruction loops. In this field, M, means mark the loop and load the loop counter while L means decrement the counter and test for the end of the loop. An explanation of all possible entries in the PGX field is given in FIGURE 4-9.

FIGURE 4-10 illustrates the use of the OPX and PGX fields for various kinds of single and multi-instruction program loops. The left-most columns indicate the instruction fields necessary to form the loop. The right-hand column indicates the instruction indexing which might be used within a loop. Single instruction loops do not use the PGX field. With either type of loop, up to two levels of looping can be provided with no requirement

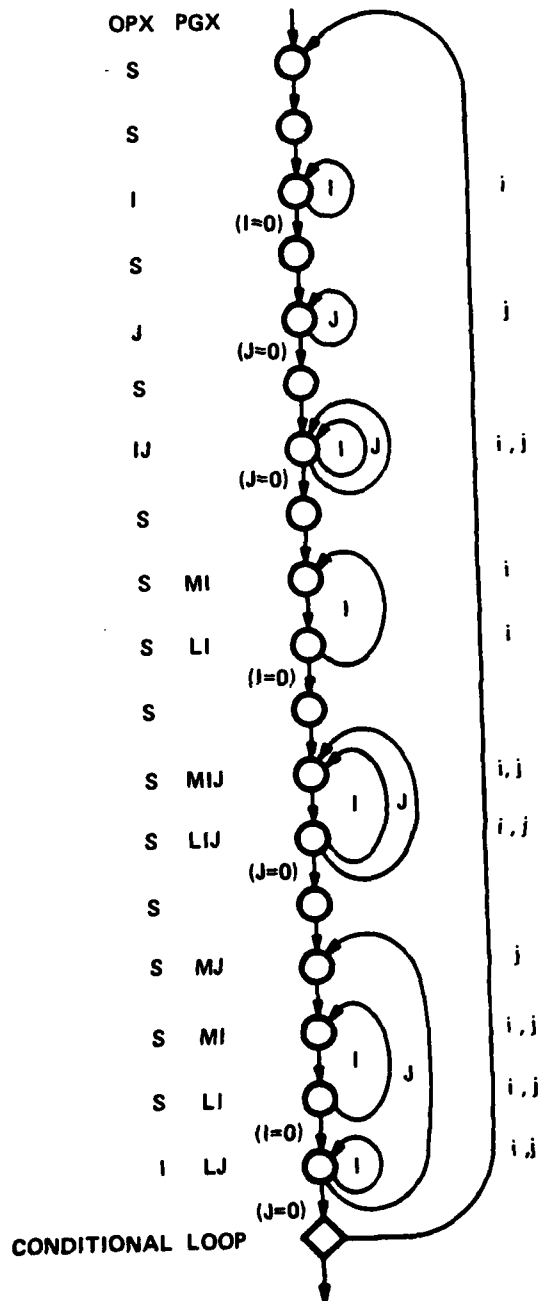


Figure 4-10. Examples of Possible Program Loops

for additional program steps for incrementing, testing, and jumping. Thus, highly repetitive operations can be performed with no time lost in the loop for loop control. This means that for convolution, correlation, or AMDF computations, the basic cycle time of 1.0 microseconds is precisely the time required for each repetition of the operation.

#### 4.3.6 A Typical PAP Program

Figure 4-11 is a flow chart for computing the correlation coefficients which are used in the Markel version of the LONG-BRAKE analyzer. The coding for this program is given in Figure 4-12. The first step in the PAP program is to set the I and J counters to 102 and 11, respectively. This controls the number of times all I and J loops will be repeated until these numbers are modified by another SET instruction.

The next step in the program is to copy the data in the speech register, X, into a second register, Y. This is done by a single move instruction, repeated I times with both X, the source, and Y, the destination arrays indexed (shifted) on each repetition. The combination of the SET and MOVE instructions require 103 microseconds execution time.

The main loop of the program is controlled by two additional instructions. The first is a MAC ( or MACD) instruction with X and Y as the source registers and C the destination. This instruction marks the beginning of a J loop and is repeated I times for each repetition of the J loop. The X and Y arrays are indexed with each repetition of this instruction and the C array is indexed each time the I count goes to ZERO. This instruction is the one which performs

$$C_J = \sum_{I=1}^{102-J+1} X_I X_{I+J}, \quad J = 1, 2, \dots, 11$$

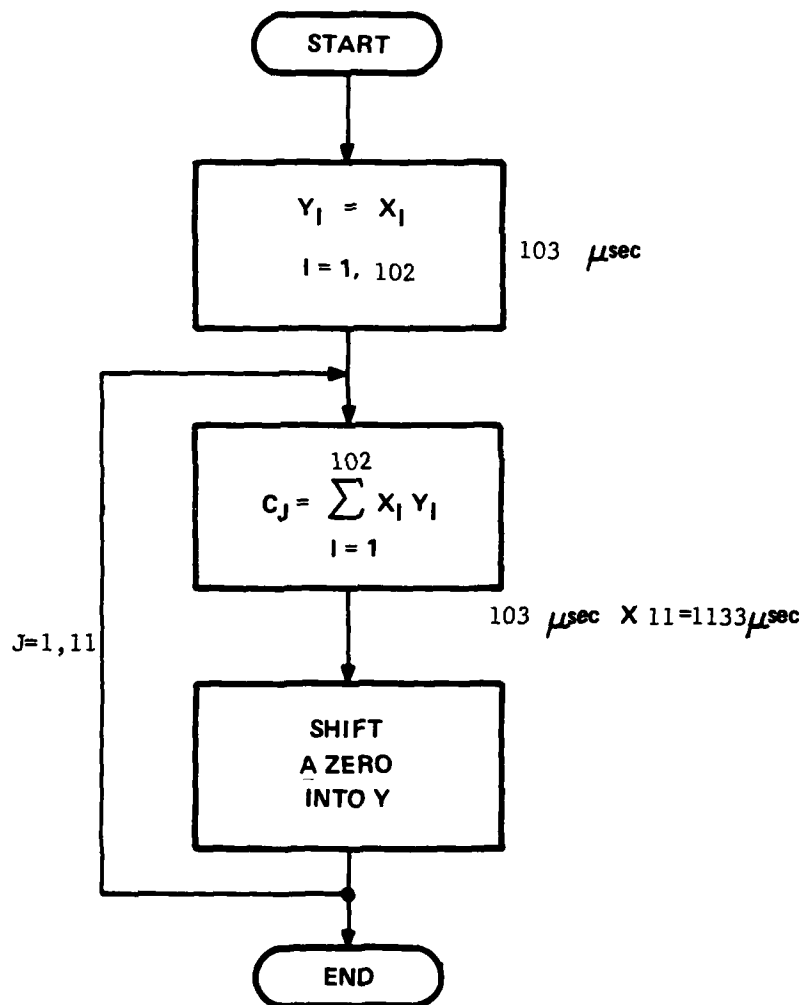


Figure 4-11. Example Computation of Correlation Coefficients

	OPCODE, OPX	S1,X	S2,X	D,X	PGX
START:	SET I	102			
	SET J	11			
	MOVE, I	X,I		Y,I	
	MACD, I	X,I	Y,I	C,J	MJ
END:	CLR, S			Y,I	LJ

FOR COMPARISON

NUMBER OF INSTRUCTIONS ON CSP-30  $\approx$  15  
 RUNNING TIME ON CSP-30  $\approx$  3.4 MILLISECONDS

FIGURE 4-12. Coding for Computation of Correlation Coefficients

the bulk of the processing and for a conventional machine would require 3 or more microseconds per repetition.

The final instruction is a CLR (or MOVE 0) instruction with the Y array as the destination. The instruction is repeated only once per repetition of the J loop and includes a shifting of the Y array with each repetition. It is this instruction which introduces the relative delay between the two sets of data. This instruction also marks the end of the J loop by virtue of the LJ command in the OPX field. As can be seen, decrementing J, testing for  $J = 0$ , and jumping back to the beginning of the loop are all included in this instruction and cost no additional time.

The total time required to perform the operations of the main loop is 1.133 milliseconds. To perform the same operations on the CSP-30 with double word accumulation would require approximately 16 instructions and 3.4 milliseconds running time.

## SECTION 5

### PERFORMANCE EVALUATION TASK REPORT

#### 5.1 Description of Tests

Tests were conducted with random bit errors for the 2400, 3600, and 4800 bit per second transmission systems using the Atal algorithms. Bit errors were introduced in a controlled manner by programming a psuedo-random bit error generator to run on the EDM speech processor as part of the operating system. The bit error generator produced a different Gaussian psuedo-random variable for each bit to be transmitted. By reversing the bit to be transmitted each time the corresponding random number exceeded a pre-determined, threshold, any desired bit error rate could be simulated. The random numbers were essentially independent, which means that no attempt was made to simulate burst errors.

Each of the three systems was evaluated at error rates of  $10^{-4}$ ,  $10^{-3}$ ,  $10^{-2}$ , and 5 times  $10^{-2}$  average errors per bit. The 2400 bps system had no error detection and correction and therefore provided an indication of the effects of transmission errors on the basic LONGBRAKE system. The 3600 bps system transmitted 61 information bits per frame protected by 20 E.D. & C bits. The E.D. & C. bits were allocated as explained in Section 2.3.2 and could only provide limited protection of the most important parameters. The 4800 bps system transmitted 61 information bits per frame protected 47 E.D. & C. bits. At this bit rate the four most significant bits of all

parameters were protected.

## 5.2 Test Results

The bit error tests have been run with several different sets of input sentences and at several different times. The results have been recorded and at least two different test recordings have been delivered to the COTR. The following is an attempt to describe the qualitative results of the tests as performed on the final delivered system. All references to intelligibility are subjective and are relative to the intelligibility of the same system with no bit errors.

### 5.2.1 Test at 2400 Bits Per Second

At 2400 bps no degradation was observed at a  $10^{-4}$  bit error rate.

At a  $10^{-3}$  bit error rate a slight increased warbling could be heard in the formants with an occasional beep or tweeting sound audible at about 5 second average intervals. The intelligibility at this error rate was not noticeably degraded.

At a one percent bit error rate the intelligibility was reduced and there were numerous relatively low level beeping and tweeting sounds in the background. There was also a moderate amount of distortion of the formants. Communication at this error rate would not be difficult, however.

At a five percent bit error rate, the speech was barely intelligible, there were numerous high level beeping and tweeting sounds and the formants were severely distorted. Communication at this error rate would be very difficult, but not impossible.

### 5.2.2 Tests at 3600 Bits Per Second

At 3600 bps no degradation could be observed at a  $10^{-4}$  bit error rate.

At a  $10^{-3}$  bit error rate very low level disturbances could occasionally be heard at about 5 second average intervals.

At a one percent bit error rate there was no noticeable loss of intelligibility but a constant low level background disturbance could be detected.

At a five percent bit error rate the system had only fair intelligibility. There were, however, constant annoying background disturbances, and distortion and warbling in the formants. Communication at this error rate would not be difficult, however.

### 5.2.3 Tests at 4800 Bits Per Second

At 4800 bps no degradation could be heard at a  $10^{-4}$  bit error rate.

A a  $10^{-3}$  bit error rate very low level disturbances could occasionally be heard at about 5 second intervals.

At a one percent bit error rate there was no noticeable loss of intelligibility but there were slight audible disturbances or distortions every few seconds.

At a five percent bit error rate, the system still had good intelligibility, but there was constant, moderately annoying low level distortion and warbling in the formants with an occasional loud disturbance. Communication at this error rate would not be difficult.

## APPENDIX A

### PRELIMINARY PAP - INSTRUCTION SET

#### ARITHMETIC LOGICAL - TWO SOURCES

<u>ADD</u>	S1 TO S2, SID*
<u>AND</u>	S1 WITH S2, SID
<u>AMDF</u>	S1 WITH S2, SID, $DEST = \sum  s1 - s2 $
<u>IOR</u>	S1 WITH S2, SID
<u>MUL</u>	S1 BY S2 (FRACTIONAL 1 WORD ROUNDED), SID
<u>MULD</u>	S1 BY S2, SID, (FRACTIONAL 2 WORD - DEST IS DOUBLE PREC. ARRAY)
<u>MAC</u>	MUL S1 BY S2 AND ACCUM (FRACTIONAL 1 WORD ROUNDED), SID
<u>MACD</u>	MUL S1 BY S2 AND ACCUM (FRACTIONAL 2 WORD - DEST IS D.P. ARRAY), SID
<u>SUB</u>	S1 FROM S2, SID
<u>XOR</u>	S1 WITH S2, SID

\*SID MEANS STORE IN DEST

#### CONTROL INSTRUCTION, PARTIAL LIST

<u>NOP</u>	
<u>HALT</u>	
<u>MMC</u>	MOVE S1 TO CONTROL WORD
<u>MCM</u>	MOVE CONTROL WORD TO DEST
<u>RSFF</u>	
<u>RSIO</u>	

## APPENDIX A - CONT

SET MOVE LITERALS TO I AND J COUNTERS

ZERO CLEAR MEMORY ARRAY CONTENTS AND LENGTH TO ZERO

## OPERATIONS ON A SINGLE SOURCE

ARS            ARITH RIGHT SHIFT S1 BY NO OF BITS SPECIFIED IN S2, SID

```
CLR      MOVE ZEROES TO DEST
```

COMP      1'S COMPLEMENT S1, SID

DARS DOUBLE ARITH RIGHT SHIFT S1 AND S2 (S1 IS HIGHER ORDER WORD, S2 IS LOWER ORDER), SHIFT BY 1 BIT ONLY, STORE IN D.P. DEST

```

LLS      LOGICAL LEFT SHIFT S1 BY NO OF BITS SPECIFIED IN S2,
SID

```

DLRS DOUBLE LOGICAL RIGHT SHIFT S1 AND S2 (S1 IS HIGHER ORDER WORD), SHIFT OF 1 BIT ONLY, STORE IN D.P. DEST

NEG 2'S COMPLEMENT S1, SID

NLLS      NORMALIZE LOGICAL LEFT SHIFT S1, STORE RESULT IN  
D.P. DEST, SHIFTED WORD IN HIGHER ORDER PART, SHIFT  
COUNT IN LOWER ORDER PART

## PROGRAM TRANSFER INSTRUCTIONS

JUMP TO LITERAL

CALL        STORE P.C. IN STACK AND JUMP TO LITERAL

RETURN POP STACK TO P.C. AND PROCEED

```
SGT      SKIP IF S1 GREATER THAN S2
```

```
SEQ      SKIP IF S1 EQUAL S2
```

**SNEQ**      **SKIP IF S1 NOT EQUAL S2**

SGE            SKIP IF S1 GREATER THAN OR EQUAL TO S2

SCT            SKIP IF CONDITION TRUE

SCF            SKIP IF CONDITION FALSE

CONDITIONS ARE OVFL, CARRY

## AND I/O CONDITIONS

- Speech Coding
  - Coding Methods
    - Vocoders
      - Adaptive Coding
        - Predictive Coding


[illegible]

**Security Classification**

Security Classification

## DOCUMENT CONTROL DATA - R &amp; D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Philco-Ford Corporation 3900 Welsh Road Willow Grove, Pennsylvania 19090		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
		2b. GROUP N/A FD-413290	
3. REPORT TITLE  LONGBRAKE II			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Final Report Covering Period 1 July 1973 to 8 November 1974			
5. AUTHOR(S) (First name, middle initial, last name) Joseph W. Foran, John P. Janowski, Gilbert M. Kaufman, Charles S. Klayman John L. Robinson, John R. Welch			
6. REPORT DATE 22 November 1974		7a. TOTAL NO. OF PAGES 102	7b. NO. OF REFS 0
8a. CONTRACT OR GRANT NO. DAAB03-74-C-0098		8b. ORIGINATOR'S REPORT NUMBER(S) Philco-Ford Contract No. 2090	
b. PROJECT NO.			
c.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d.			
10. DISTRIBUTION STATEMENT 			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY National Security Agency 9800 Savage Road Fort George G. Meade, Md.	
13. ABSTRACT Longbrake II is a 12 month program consisting of both study and hardware phases. The objective of the study is to develop an optimum Linear Predictive Coding vocoder system by optimizing the current real time algorithms used, and by the evaluation of additional LPC related techniques. The purpose of the hardware effort is to develop, test, and evaluate two Exploratory Development Models (EDM) of a speech compression system which incorporates the Atal and Markel approaches to adaptive linear estimation.  The work accomplished in the final quarter is summarized, a description of system performance over channels with bit errors is included, and a proposed ADM hardware design is described.			

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

